# Human-Machine Collaborative Planning

**James Allen and George Ferguson**

Department of Computer Science
University of Rochester
Rochester, NY, USA 14627-0226
{james,ferguson}@cs.rochester.edu

**Abstract**

This paper describes the design of computer agents that can collaborate with humans in planning. It includes an explicit problem solving level that mediates between the human-computer interaction and the underlying automated plan reasoners. We also describe a plan reasoning system that allows for incremental, interactive development of plans to support collaborative planning. This model has been used in a prototype system in which untrained users can successfully develop plans in a simple evacuation-planning domain.

## Introduction

In many space applications, there is a great need for automated techniques that can support human planning and decision-making. This requires taking a new approach to automated plan reasoning to support the type of interaction that is most useful for humans. We think that this activity is best viewed as a collaboration between the human and machine similar to how humans collaborate to solve problems. As with human collaboration, both participants bring their own special skills to the process. By having humans and machines work together, taking advantage of each other's strengths, we will be able to solve harder problems and find better solutions than either could find working alone. Even if a person could ultimately produce the same quality plan, the collaborative human-machine process will be faster.

Collaborative planning requires capabilities often not found in traditional planning systems. Most important, the development of plans must be *incremental*—allowing people to develop plans by focusing on a small part of the plan, exploring options, and making a few decisions before considering the rest of the problem. Effective interactive planning requires *plan stability*–which means that plans should only minimally change when new constraints are added. A system that computes a new optimal plan at each step may cause the entire plan to radically change at each interaction, causing confusion and frustration to the human participant. In addition, planning must be accomplished *without a fixed search strategy*—there is no simple systematic order (e.g., top-down, bottom-up) in the choice of subproblems the human may chose to focus on next. Finally, the system must be *open to innovation*—it must be able to build, under human direction, plans it otherwise could not produce and be able to validate and reason about such novel plans.

We have built a prototype of just such a system in which a user collaborates with the system to build plans to evacuate the inhabitants of a fictitious island (Ferguson & Allen, 1998). The framework described here is a specific instantiation within a larger architecture for robust, dialogue-based mixed-initiative interaction (Allen, Ferguson, & Blaylock, 2002; Allen et al, 2001). We will not dwell on the general architecture here, but rather focus on it as instantiated for collaborative planning.

There are several contribution in this paper. The first is the specification of a **domain-independent problem-solving level** that mediates between the communication management and the actual reasoning capabilities of the system. The second is the specification of **a hybrid plan reasoning system** that supports the needs of the human collaborator. We believe that far more attention needs to be paid to the gap between the abilities of automated reasoners and the needs of human decision makers. The architecture we propose bridges this gap and provides the foundation for systems that can significantly enhance human performance.

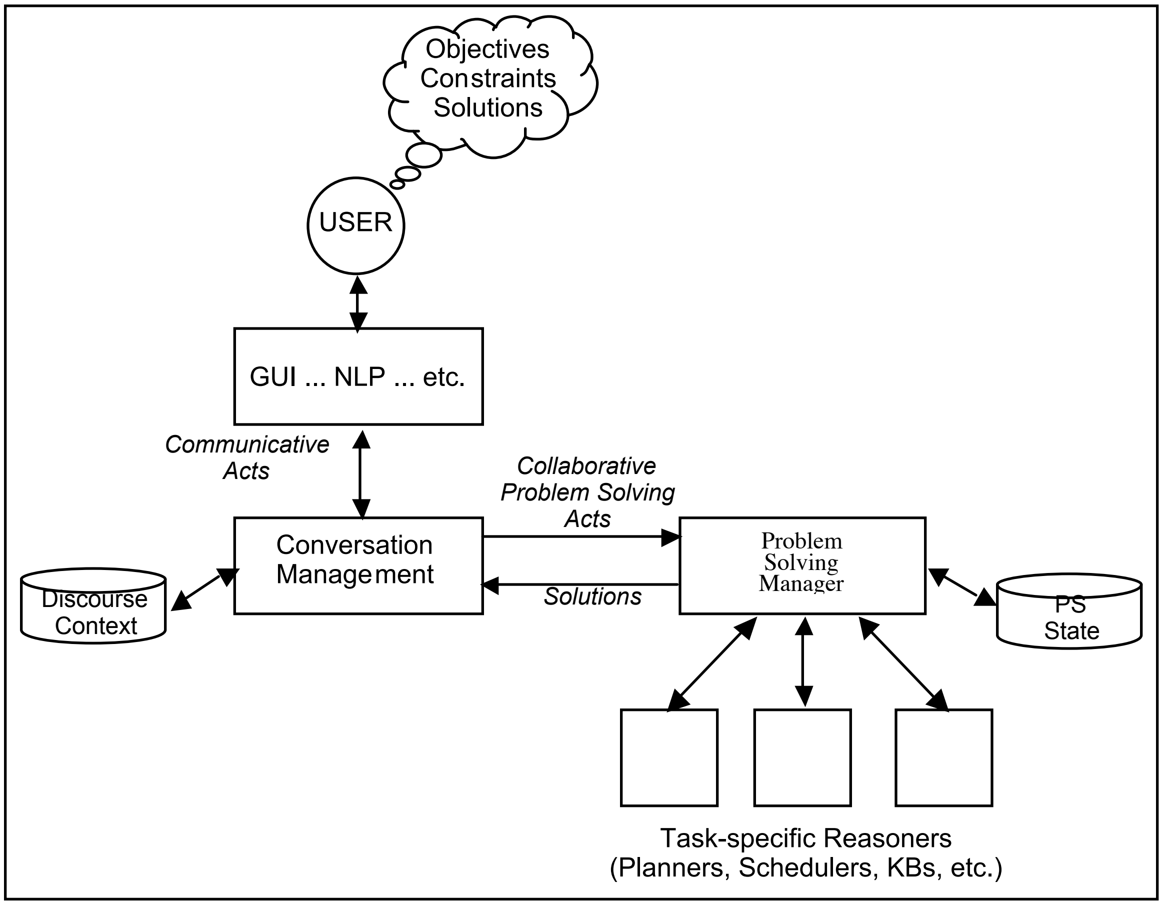


**Figure 1: Abstract view of collaborative problem solving architecture**

## Collaborative Problem Solving

Figure 1 gives a high-level view of our approach. Humans have objectives that they wish to achieve, constraints on possible solutions, and possibly solutions or parts of solutions to these problems. These are not, of course, directly available to the computer agent, but rather are received via some kind of human-computer interface. In our current system, we use a multimodal interface combining natural language with graphical displays. The output of the language processing components is a series of communicative acts that the system must respond to. These acts are transformed into collaborative problem solving acts that capture what the human is trying to do[1].

The problem solving level maintains the state (and history) of the collaborative problem solving process. Using this state, it provides services to help the conversation manager identify the intended interpretation, and it then coordinates the invocation of the specialized plan reasoners. The specialized reasoners themselves are closest to traditional AI technologies such as planners, schedulers, knowledge bases, and the like. The architecture supports the use of such components on a "plug-and-play" basis, the idea being that specific components can be assembled for a specific problem-solving task, possibly dynamically as the problem solving proceeds.

Traditional planning representations and algorithms, however, are often not directly suitable for use in incremental, user-centered collaborative planning. A planner that simply finds complete plans given goals is not going to be much use when the user wants to specify objectives incrementally, adding and removing constraints

[1] Of course, not all communication is about plans (e.g., consider greetings or clarifications of intent). Such acts are ignored here as they are not the focus of this paper.

and suggesting partial solutions as the plan is developed.

The remainder of this paper describes our approach in more detail. The next section describes the representation of the problem solving state maintained at the problem solving level. This is followed by a description of the tasks performed by the problem solving level and their interaction with the specialized reasoners. We then describe the specific planning model we developed to handle incremental interactive planning, and finally conclude with a discussion of related work and open issues.

## An Overview of the Problem Solving State

One of the main functions of the problem solving manager (PSM) is the maintenance of the collaborative problem solving state and the history of problem solving. We have developed a four-level representation of plans that combines features from several branches of AI planning to meet the needs of collaborative problem solving. The four levels are:

- **Hierarchical Objectives**: This is an abstract specification of the objectives (i.e., goals) being pursued. Objectives can be broken down into sub-objectives with constraints between them. This level of representation provides much of the context for interpreting new interactions, and supports HTN-like algorithms for goal refinement and elaboration.
- **Task Structure (or abstract plan):** This level captures abstract solutions being considered for the objectives. Constraints on possible courses of action are maintained together with causal connections and independence assumptions. These abstract solutions are a compact summary of a class of possible concrete solutions under consideration.
- **Scheduled COA or "Straw Plan"**: This level captures a particular concrete course of action that satisfies the constraints in the abstract plan (produced by committing to various choice options and scheduling the actions based on their expected durations). This nominal straw plan is often presented to the user to ground the interaction with a concrete proposal.
- **Expected Scenario**: This is a representation of the state of the world during the projected execution of the plan, also including exogeneous events and there effects if appropriate. It is built from abstract simulations operating on the straw plan.

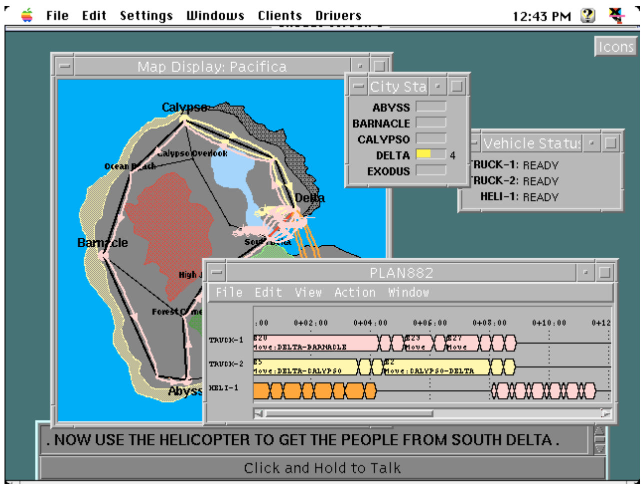
The four levels capture different aspects of plan-related knowledge that are necessary to support effective interaction. The objective hierarchy captures the goals driving the interaction. The human often adds or removes constraints that affect what solutions are possible, and the system uses HTN-based algorithms to decompose objectives into tasks based on these constraints. The tasks are instantiated into an abstract plan using a domain-independent algorithm (the plan establisher) that uses algorithms derived from temporal logic planning (Allen & Koomen, 1983; Allen et al., 1991). The abstract plan provides much of the structure for intelligent plan revision. The abstract plan is used to produce a straw plan using routing and scheduling components. The straw plan is used to generate many of the maps and charts presented to the human. Feedback about the straw plan from the human typically produces additional constraints at the abstract plan or objective levels, and then a new straw plan can be generated. Finally, the expected scenario provides a temporal representation of the state of the world before, during and after the plan, and is used to support the user's evaluation and exploration of the expected impact of the plan.

## Managing the Problem Solving Interaction

One of the main requirements on the PSM is to support the interpretation of user interactions, relative to the current problem solving state. In particular, it must

1. Help determine what problem solving operation the user intended; and
2. Identify which aspects of the plan are being modified and how.

It is important to realize that although we discuss these two tasks separately, performing one of them properly generally requires performing the other. We often cannot be sure what problem solving act is intended unless we can identify what parts of the plan may be affected by the operation. In reverse, we often can't identify what parts of the plan are being affected until the in-



. NOW USE THE HELICOPTER TO GET THE PEOPLE FROM SOUTH DELTA .

**Figure 2: Evacuating an island**

tended problem solving operation is identified. Thus, possible interpretations arise by considering each of these problems simultaneously and considering the constraints that an interpretation at one level imposes on interpretations at other levels.

## Identifying Problem Solving Operations

There are a wide range of problem solving operations that can be intended in an interaction. It will help to explore these in a more concrete setting. Consider the Pacifica domain in which a person must work with the system to plan the evacuation of residents from an island in the face of an oncoming hurricane. Figure 2 shows a screen shot with a session in progress. Shown is (1) a map displaying the routes planned so far; (2) a time-line chart displaying the current plan; (3) a "city status" chart that shows where the remaining people are; (4) a "vehicle status" chart that shows the availability of each vehicle; and (5) the output of the speech recognition from the human's last command.

Some interactions make the problem solving operation fairly explicit in the user's input, as in "Delete the plan with the helicopter," "Compare those two options," or "Simulate the plan and give me the expected arrival time of the crew." In these, the action described is the problem solving operation to be performed. In other cases, the problem solving operation is not explicit and the user focuses on the content of the plan. For instance, the question "Can we use a helicopter to get the people from Abyss" is ambiguous, being either a question about capabilities or a suggestion of a course of action. Independent of this decision, there are three possible problem solving operations that might be being queried or suggested. The user might be:

1. **Suggesting** a new goal that was not present before, and may require further elaboration to arrive at a solution.
2. **Extending** an existing solution. Say we need to get the people to Delta from Calypso, and we have just moved them by truck to Abyss. The proposed action will then extend and complete the development of the solution to this goal.
3. **Modifying** an existing solution. Say we have already a plan to move the people from Abyss to Delta by truck. This is then asking about the possibility of moving by helicopter instead, or suggesting that we do so.

There are a number of techniques that are used to identify the intended problem solving operation. Some of these depend on the communicative act. For instance, one key factor for requests and suggestions is the feasibility of actually performing the proposed problem solving operation. An example of this is that it would be incoherent to introduce a new goal that is either already present or is subsumed by the existing goals in the plan. The problem solving operation is identified by considering the following factors:

1. Lexical and syntactic/semantic cues: certain words signal the intended operation (e.g., "instead" signals a modification of some sort, "I want to do *X*" makes the introduction of a goal the most likely interpretation although others remain possible). These are handled at the communicative level and passed on to the problem solving level.
2. Plausibility tests dependent on the problem solving operation (e.g., introduced goals cannot exist already, to modify a plan by deleting an action, the plan must contain that action, etc.).

| | |
|---|---|
| We need to get the people from Exodus to Delta for evacuation ASAP | Introduce an objective |
| Let's use a helicopter | Refine a goal |
| No, let's use a helicopter from Bath instead | Modify/Correct solution |
| Scratch that | Undo operation |
| How long will it take | Evaluate plan |
| Oh, let's take them to Calypso for evacuation instead | Modify goal |
| First, fly them to high pass junction | Specify solution |
| Then use a truck to get them to Calypso | Extend solution |
| What if we used some buses instead | Compare options/solutions |
| Forget it | Reject option/solution |
| Let's use the bus plan then | Select option/solution |
| Oh, the weather's changed, forget the whole thing | Cancel plan |

**Figure 3: Sample dialogue illustrating common problem solving operations (system would respond with a plan after each statement)**

3. Preferences derived from a model of human problem solving model: Eventually we want a rich model of how people solve problems and track their actions through this model to predict plausible next steps in the problem solving process. Currently, we use a simple preference strategy first suggested in (Litman & Allen, 1987). In the absence of lexical cues that will change the ordering, we prefer interpretations that extend the current solution first, then interpretations that modify it, and only consider the introduction of new goals as a last resort.

The main problem solving actions we have encountered in our planning domains are illustrated by the somewhat whimsical hypothetical dialogue shown in Figure 3, where you can imagine the system responding with a modified plan at each stage.

## Identifying the intended plan and situation

Since the user may be considering alternate ways to accomplish their objectives, possibly with different underlying assumptions or specifications, there may be multiple plans and possible situations under consideration. The problem solver must identify which of these is being affected. In addition, different human-computer interfaces will provide different means for the user to refer to their plans.

So far, we have found reasonable success using a strong focusing assumption in which we expect the user to continue working on the same plan in the same situation unless they explicitly do something that indicates otherwise. We keep track of what (sub)objective was last discussed and, when there is no evidence indicating otherwise, we attempt to interpret the next interaction with respect to the same objective. Of course, when the user explicitly changes focus of attention by saying something like “Let’s go back to the helicopter plan” we change the focus as indicated. In addition, they may invoke more complex operations such as comparisons, such as by saying “Which option is faster?” The current prototype’s capabilities in this area are fairly minimal and only limited option comparisons are supported.

## Identifying range and type of modification

Having identified the relevant plan and situation, the next step is to identify the specific objective or part of the solution that concerns the user. We use focusing heuristics again and expect the user to continue to work on the subgoal previously discussed unless there is some evidence to the contrary. Each candidate operation is associated with a set of well-formedness filters that must be satisfied for the interpretation to make sense in the current context.

An example of this filtering process is the way the problem solver treats modification requests. For such requests, three things are derived from the analysis of the input and context: (1) a set of necessary conditions on the old goal or solution, (2) parts of the plan to delete, and (3) parts of the plan to add. The necessary conditions on the old plan are essential for preventing unintuitive and incoherent plan modifications.

For instance, consider the command "Send truck one to Abyss instead of Bath." For this command, we would derive the following information:

**Necessary conditions**: the relevant solution involves an action involving truck one and the destination Bath;

**Deletes**: destination Bath

**Adds**: destination Abyss

When searching for a plausible subplan to modify, the system will search according to its focus heuristics looking for a subplan that satisfies the necessary conditions.

To perform the actual modification, it must determine whether the intention is to modify the goal or the solution. This is determined by checking whether the delete condition is part of the objective at the top level of the current plan. If it is, then the objective is modified and the planner is invoked to find a new solution. If the destination is not part of the objective, then it must be part of the particular solution at hand. The problem solver finds all actions in the solution that involve the truck going to the destination, and instructs the planner to delete those actions, insert new actions with the new destination, and replan to make the overall plan coherent again.

## Examples of Problem Solving Operations

We can illustrate the operation of the problem solver and its interaction with the plan reasoners by considering the problem solving operations in Figure 3 and their effect on different levels of the plan representation (problem solving state). We claim that all four levels of this plan representation are necessary to support collaborative planning. This will be seen by considering how the problem solving operations affect different levels of the plan representation.

**Introduce/Refine objective**: The introduction or refinement of an objective results in an addition to the top-level hierarchical objectives. If it is sufficiently specified, the system refines it into tasks, and so on down to a scheduled COA. Thus, the effects of introducing a goal trickle down to other levels.

**Modify/correct goal or solution**: As noted previously, these operations are almost always ambiguous between whether they are modifying the goal (at the top level) or modifying an aspect of the current solution (at the task level). It is important, however, that the problem solver get the level right. For example, a request to "use a helicopter instead" seems like a direct modification of a role in one or more actions in the plan. However, if a plan for transporting cargo using a helicopter is qualitatively different from, say, using a truck (perhaps due to capacity constraints), then any attempt to manipulate the solution directly will result in incoherent plan. Instead, the goal specification must be modified, then that part of the solution replanned.

**Evaluate plan**: This operation involves the plan representation at different levels. The evaluation itself is done using either the expected scenario or the scheduled COA, depending on the type of evaluation (the former being less committed). But the evaluation criteria comes from the objectives of the plan, such as the required time of completion (whose constraints are derived from expressions such as "as soon as possible" or "by ten o'clock". )

**Specify/Extend solution**: These operate at the abstract plan level by introducing specific actions to the plan. Supporting these operations requires a planner that can do more than simply create plans given goals. In our experience with the prototype system, these are the most common operations, so our planner is optimized to perform simple extensions quickly. The plan representation to support this is described in the next section.

**Create/compare/reject option/solution**: These operations operate on entire plans stored in the problem solving state, rather than on any particular level of a single plan.

**Undo operation**: Uses the problem solver's history of the problem solving state to revert to a previous point in the problem solving process. Note that most inputs that have an undo interpretation can also be interpreted as some kind of modification.

**Cancel plan**: This operation deletes both a solution (course of action) and its high-level goal(s). For example, saying "Don't send the truck to Abyss" (a Modify) would leave the goal of evacuating the people from Abyss, whereas "Forget about the people at Abyss" (a cancel) means that we no longer need to worry about that goal.

## Planning for Collaborative Problem Solving

The PSM provides the glue that connects the human-computer interface to the automated reasoners. However, this still leaves the problem of developing planners and other reasoners capable of supporting the problem solving operations recognized by the problem solving level.

Our approach to building such a planner was motivated by several considerations:

**Speed**: Since we expected to be operating in an interactive environment, the planner must be able to respond quickly, or at least be optimized for common cases arising during collaboration.

**Expressiveness**: We required a rich representation of action, plans, time, and the world in order to support both expressive interaction possible from the NLP interface, and to support reasoning about realistic domains.

**Completeness**: We were not overly concerned with the usual form of completeness for planners. We could live with the planner not always finding plans where they exist—provided that the user and the system can interact in order to iterate towards a plan. What was essential was that the planner could represent, validate, and extend a correct plan when presented with it.

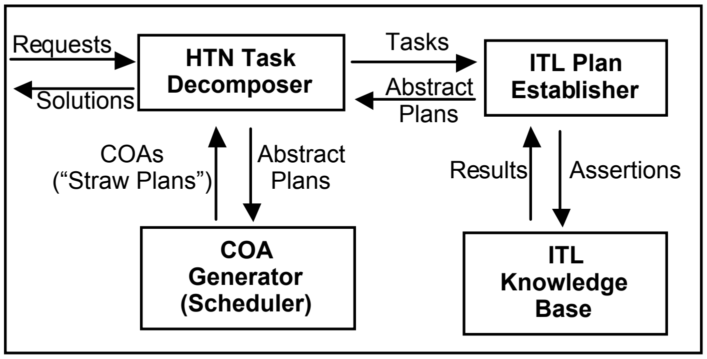


**Figure 4: The Hybrid Planning Architecture**

**Flexibility**: The planner must support a range of operations (i.e., not just finding a plan given goal) in order to support the user during collaborative problem solving.

Of these, flexibility was probably the most important, and was also the least explored in the planning literature. It turned out that we could specify a very flexible and yet still well-founded planning system by modifying and extending some traditional planning algorithms.

## A Hybrid Planning Architecture

Because of expressivity concerns, we started with the Interval Temporal Logic (ITL) planner described (Allen & Koomen, 1983; Allen et al., 1991). At an abstract level, this algorithm is in fact very similar to standard partial-order planner such as UCPOP (Penberthy & Weld, 1992). The major differences are: (1) persistence assumptions are used, rather than causal links, to record cause and effect relationships, (2) ordering constraints arise implicitly from temporal constraints on intervals rather than by being added explicitly, and (3) threats to links are discovered by forward-chaining using axioms that render incorrect plans temporally inconsistent. But the basic algorithm is the same in both approaches: (1) find an unsupported goal; (2) establish it by persistence or by adding an action; (3) resolve problems. And in fact, both algorithms are equally unsuitable for use on realistically complex problems due to their poor performance.

To improve performance but preserve the ability to reason about complex worlds using the full power of the temporal representation, we developed the planning model shown in Figure 3. The model is a **hybrid** because it involves several components that contribute different capabilities to the overall planner and that are based on different types of knowledge and reasoning.

The Task Decomposer is an HTN-like planner that decomposes objectives into tasks (sets of events with temporal constraints between them). For example, a request to evacuate all the inhabitants from one city to another is decomposed into a task involving getting vehicles there, loading the people into them, sending them to the destination, and unloading the people. There are can be several ways to decompose an objective. Like the problem solver above it, the task decomposer can consider alternative decompositions looking for the best solution.

The Plan Establisher is really the original temporal logic planning algorithm but without the ability to add events (steps) to the plan. That is, it computes persistence assumptions, maintains binding and ordering constraints, and updates the temporal database, but it relies on the decisions of the Task Decomposer as to which events are to be part of the plan. Of course, this may still involve significant amounts of search. The Plan Establisher can reason about plans involving conditional effects, simultaneous and overlapping actions, and external events.

The ITL Database maintains a description of the world described using temporal intervals with constraints between them. At the end of planning, this database provides a temporally-explicit description of the world before, during, and after the execution of the plan. This capability is essential to many problem solving operations that query the state of the world at some point during the plan.

Finally, the COA Generator produces fully explicit plans with actions scheduled at specific times consistent with the constraints of the abstract plan. It produces the "straw plans" presented to the user in order to ground the interaction in something concrete. The current scheduler uses an $O(n^3)$ approximation to the general ITL scheduling problem, which is NP-complete.

## Plan Modification

One of the most interesting challenges we faced was modifying plans. There are two types of modification: extension and substitution.

Extending a plan is similar to traditional planning for goals, except that it is **incremental**. While

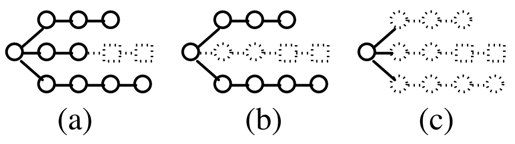


**Figure 5: The Three Modification Strategies**

most traditional planners can handle conjunctive goals, handling extensions by adding a new conjunctive goal to the plan is likely to be a costly way of handling extension and, more importantly, typically doesn't preserve the existing solution. We desire extensions that change the existing plan as little as necessary. This minimal change principle is crucial in collaborative planning. The hybrid planner is ideally suited to implementing the extension operation. Given an existing plan and a temporal context, the Task Decomposer first decomposes the new objective into a set of events. It must then also determine the appropriate context in which to call the Plan Establisher. What this means is that it can choose to keep certain aspects of the prior plan fixed by adding them to the temporal database rather than re-establishing them.

In most cases in our experience, the new actions can be added to the existing plan without significant change. We call this **monotonic extension**. There are of course cases where the assumptions necessary for the new events require that some previous assumptions be retracted and re-established. We call this **non-monotonic extension**, and it can be non-monotonic with respect to the task being extended or to the entire plan. In more detail, these three possibilities are shown in Figure 4. The circles represent events already in the plan, squares represent the actions being added. The solid outlines indicates what is kept fixed in the plan (i.e., it is added to the temporal context prior to trying to establish the remaining events).

Case (a) is monotonic extension. All previous events in the plan are added to the temporal context together with their supporting assumptions. The new events (the squares) are then passed to the Establisher to be integrated into the plan. This will only succeed if the Establisher can insert these actions into the plan without changing any existing ordering or persistence assumptions in the old plan. Note that the Establisher can insert actions between existing actions as long as the initial ordering constraints and persistence assumptions are preserved.

The remaining cases are non-monotonic. In case (b), all the events in the plan except those in the focus task (the one being extended) are added to the temporal context. Then all events in the focus task, including the new events, are passed to the Establisher. In this case, ordering and persistence assumptions between events in the focus task are recomputed, but everything else remains fixed.

Case (c) is taken only if all else fails. The new events are added to the task being extended, and then all the events in the tasks are passed to the Establisher in the initial context. In effect, this allows complete replanning and may produce a plan substantially different from the original plan.

In our experience with the prototype system, monontonic extension accounts for the vast majority of planning (extension) performed by the system.

The extension operation can be generalized into a modification operation. As noted in the description of the problem solver, a modification can be broken into (a) new objectives or constraints to add, (b) events or constraints to delete, and (c) events or constraints to modify. The hybrid planner implements modification requests by first having the Task Decomposer identify parts of the plan to remove and then constructing a temporal database reflecting the resulting situation. This is done with a combination of event definitions and heuristic rules. It then uses the extension operation of the Plan Establisher to add the new additions to the plan. Again, the goal is to attempt simple invocations of the Plan Establisher first, in the hopes of not needing to do the larger search. While the current prototype can only perform a limited class of modifications, these nonetheless account for the majority of the modification attempts during collaborative planning so far with the system.

## Related Work

Explicit models of problem solving have been used in a number of dialogue systems (e.g., (Lambert, 1991; Litman & Allen, 1987; Ramshaw, 1991)). None of these were developed into comprehensive models that could cover most of the interactions in collaborative problem solving dialogue. They also did not support interaction with state-of-the-art planning systems.

Prior work on plan modification and reuse (Kambhampati, 1994; Kambhampati & Hendler, 1992) also uses a hierarchical representation of plans, and annotates them with causal dependency links. The emphasis is on the formal specification of transformations of plans during modification, such as in retrieving plans from a case base. We have concentrated on the issue of determining the intention underlying modification in the collaborative planning situation. We might be able to apply some of the plan modification algorithms if representational differences can be overcome.

Work on advisable planning (Myers, 2000) shares many common goals with our approach as it explores how planners can build plans given a set of constraints on possible solutions. The main difference between that work and ours is the role the human plays in the process. In advisable planning, the human comes up with a set of constraints on solutions and the planner finds a plan. In our approach, the human and system collaboratively build plans in an interactive and incremental way.

## Future Work

The current implementation is robust in the sense that untrained users can interact with the prototype system to solve evacuation problems and have a good chance of constructing a useful plan that meets the goals. However, they generally do not solve these problems any faster that they would by interacting with another person who was given the same information that the system has. While significant work needs to be done on scaling up the techniques, we believe we have demonstrated that this model has great potential.

There are significant issues on the speech and language processing end that we are addressing but will not discuss here. One of the major weaknesses of the current system involves the identification of implicit high-level goals. That is, if the user suggests performing an action without ever having mentioned the goal motivating that action, it is important that we nonetheless identify the goal so that subsequent operations, such as modifications, can be properly interpreted. This is one of the major causes of failure in the current system that prevents users from modifying some plans as they wish.

Here are some additional problems that we are working on:

- Improving the ability to identify which plan is affected by proposed problem solving operations.
- Extending the notion of plan options to better support "what-if" reasoning.
- Replacing the current preference ordering of problem solving operations with a more detailed model of problem solving behavior to guide the problem solving level.
- Continuing to improve the specialized reasoners, particularly the planner, without sacrificing their flexibility and their good performance on the most common operations.

The ultimate goal of this project is to show that an untrained user collaborating with the system can solve problems faster and better than two people working together. Achieving this goal will require progress on both the architecture and the implementation. But we believe it is feasible within the next five years.

## Conclusions

We have presented an architecture for collaborative human-computer problem solving and described algorithms for key components in this architecture. We have used this model in a system that can collaboratively constructing plans in an intuitive way in a realistic domain. The problem solving level of this architecture is the glue that connects the human-computer interface to the underlying planners and reasoners. The multi-level representation of plans used by the problem solver supports both interpretation of user intention and effective construction and modification of plans.

## References

Allen, J., Ferguson, G., & Blaylock, N. (2002). A Problem Solving Model for Collaborative Agents. Paper presented at the AAMAS, Bologna, Italy.

Allen, J., & Koomen, J. A. (1983). Planning using a Temporal World Model. Paper presented at the 8th Int'l. Joint Conference on Artificial Intelligence, Karlsruhe, Germany.

Allen, J., Tenenberg, J., Pelavin, R., & Kautz, H. (1991). Formal Models of Reasoning About Plans.: Morgan Kaufmann.

Allen, J. F., Donna K. Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, Amanda Stent. (2001). Towards Conversational Human-Computer Interaction. AI Magazine, 22(4), 27-35.

Ferguson, G., & Allen, J. (1998). TRIPS: An Integrated Intelligent Problem-Solving Assistant. Paper presented at the NCAI (AAAI-98), Madison, WI.

Kambhampati, S. (1994). Exploiting causal structure to control retrieval and refitting during plan reuse. Computational Intelligence, 10(2), 212-245.

Kambhampati, S., & Hendler, J. (1992). A validation-structure-based theory of plan modification. Artificial Intelligence, 55, 193-258.

Lambert, L. & Carberry, S. (1991). A Tripartite Plan-Based Model of Dialogue. Paper presented at the Proc. of Association for Computational Linguistics.

Litman, D., & Allen, J. (1987). A Plan Recognition Model for Subdialogues in Conversation. Cognitive Science, 11(2), 163-200.

Myers, K. Planning with Conflicting Advice. In Proc. Of 5'th Intl Conf. On AI Planning and Scheduling (AIPS-2000),

Penberthy, J. S., & Weld, D. S. (1992). UCPOP: A Sound, Complete, Partial Order Planner for ADL. Paper presented at the Third Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR '92), Boston, MA.

Ramshaw, L. A. (1991). A three-level model for plan execution. Paper presented at the 29'th Annual Mtg of the Association for Computational Linguistics (ACl-91), Berkeley, CA.