A METAPLAN MODEL FOR PROBLEM-SOLVING DISCOURSE*

Lance A. Ramshaw
BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02138 USA

ABSTRACT

The structure of problem-solving discourse in the expert advising setting can be modeled by adding a layer of metaplans to a plan-based model of the task domain. Classes of metaplans are introduced to model both the agent's gradual refinement and instantiation of a domain plan for a task and the space of possible queries about preconditions or fillers for open variable slots that can be motivated by the exploration of particular classes of domain plans. This metaplan structure can be used to track an agent's problem-solving progress and to predict at each point likely follow-on queries based on related domain plans. The model is implemented in the Pragma system where it is used to suggest corrections for ill-formed input.

1. INTRODUCTION

Significant progress has been achieved recently in natural language (NL) understanding systems through the use of plan recognition and "plan tracking" schemes that maintain models of the agent's domain plans and goals. Such systems have been used for recognizing discourse structure, processing anaphora, providing cooperative responses, and interpreting intersentential ellipsis. However, a model of the discourse context must capture more than just the plan structure of the problem domain. Each discourse setting, whether argument, narrative, cooperative planning, or the like, involves a level of organization more abstract than that of domain plans, a level with its own structures and typical strategies. Enriching the domain plan model with a model of the agent's plans and strategies on this more abstract level can add significant power to an NL system. This paper presents an approach to pragmatic modeling in which metaplans are used to model that level of discourse structure for problem-solving discourse of the sort arising in NL interfaces to expert systems or databases.

The discourse setting modeled by metaplans in this work is expert-assisted problem-solving. Note that the agent's current task in this context is creating a plan for achieving the domain goal, rather than executing that plan. In problem-solving discourse, the agent poses queries to the expert to gather information in order to select a plan from among the various possible plans. Meanwhile, in order to respond to the queries cooperatively, the expert must maintain a model of the plan being considered by the agent. Thus the expert is in the position of deducing from the queries that are the agent's observable behavior which possible plans the agent is currently considering. The metaplans presented here model both the agent's plan-building choices refining the plan and instantiating its variables and also the possible queries that the agent may use to gain the information needed to make those choices. This unified model in a single formalism of the connection between the agent's plan-building choices and the queries motivated thereby allows for more precise and efficient prediction from the queries observed of the underlying plan-building choices. The model can be used for plan tracking by searching outward each time from the previous context in a tree of metaplans to explore the space of possible plan-building moves and related queries, looking for a predicted query that matches the agent's next utterance. Thus the examples will be presented in terms of the required search paths from the previous context to find a node that matches the context of the succeeding query.

This metaplan model is discussed in two parts, with Section 2 covering the plan-building class of metaplans, which model the agent's addition of new branches to the domain plan tree and instantiation of variables, while Section 3 presents examples of plan feasibility and slot data query metaplans, which model the agent's strategies for gathering information to use in

*This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-85-C-0016. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

plan-building. Section 4 then compares this modeling approach to other plan-based styles of discourse modeling, Section 5 discusses applications for the approach and the current implementation, and Section 6 points out other classes of metaplans that could be used to broaden the coverage of the model and other areas for further work.

2. PLAN BUILDING METAPLANS

In this approach, the plan-building metaplans discussed in this section model those portions of problem-solving behavior that explore the different possible refinements of the plan being considered and the different possible variable instantiations for it. The domain for all the examples in this paper is naval operations, where the agent is assumed to be a naval officer and the expert a cooperative interface to a fleet information system. The examples assume a scenario in which a particular vessel, the Knox, has been damaged in an accident, thereby lowering its readiness and that of its group. The top-level goal is thus assumed to be restoring the readiness of that group from its current poor rating to good, expressed as (IncreaseGroupReadiness Knox-group poor good).

The domain plans in Pragma are organized in a classification hierarchy based on their effects and preconditions, so that a node in that hierarchy like the top-level instance of IncreaseGroupReadiness in the examples actually stands for the class of plans that would achieve that result in a certain class of situations. The plan class nodes in this hierarchy can thus be used to represent partially specified plans, the set of plans that an agent might be considering that achieves a particular goal using a particular strategy. The subplans (really plan subclasses) of IncreaseGroupReadiness shown in Figure 1 give an idea of the different strategies that the agent may consider for achieving this goal. (Variables are shown with a prefixed question mark.)

```
(IncreaseGroupReadiness
    Knox-group poor good)                    (1)
  (RepairShip Knox)                          (2)
  (ReinforceGroup Knox-group ?new-ship)      (3)
  (ReplaceShip Knox ?new-ship)               (4)
```

Figure 1: Subplans of IncreaseGroupReadiness

The plan classification depends on the circumstances, so that RepairShip only functions as a subplan of IncreaseGroupReadiness when its object ship is specified as the Knox, the damaged one, but some of the plans also introduce new variables like ?new-ship, introduced by the ReplaceShip plan, that can take on any value permitted by the plan's preconditions. Each of these plans also has its own subactions describing how it can be achieved, so that ReplaceShip, for example, involves sailing the ?new-ship to the location of the damaged ship, having it take over the duties of the damaged ship, and then sailing or towing the damaged one to a repair facility. Those subactions, in turn, specify goals for which there can be multiple subplans. The metaplan structures modeling the problem-solving discourse are built on top of this tree of domain plans and actions.

Plan Refining Metaplans

The *build-plan* metaplan is used to capture the agent's goal of constructing a plan to achieve a particular goal, with the *build-subplan* and *build-subaction* metaplans modeling the problem-solving steps that the agent uses to explore and refine the class of domain plans for that goal. An instance of *build-subplan*, say, reflects the agent's choice of one of the possible subplan refinements of the current domain plan as the candidate plan to be further explored. For example, the initial context assuming an IncreaseGroupReadiness plan due to damage to the Knox would be represented in our model by the *build-plan* node on line (1) of Figure 2.

```
(build-plan
   (IncreaseGroupReadiness
      Knox-group poor good))                 (1)
 (build-subplan
    (IncreaseGroupReadiness ...)
    (ReplaceShip ...))                       (2)
  (build-plan
     (ReplaceShip Knox ?new-ship))           (3)
   (build-subaction
      (ReplaceShip ...) (Sail ...))          (4)
    (build-plan
       (Sail ?new-ship ?loc Knox-loc))       (5)
```

Figure 2: *Build-Plan*, *Build-Subplan*, and *Build-Subaction*

If we suppose that the agent first considers replacing Knox with some other frigate, that would be modeled as a *build-subplan* child (2) of the *build-plan* for the IncreaseGroupReadiness plan (1), that would in turn generate a new *build-plan* for ReplaceShip (3). If the agent continues by considering how to get the new ship to that location, that would be represented as a *build-subaction* child (4) of the *build-plan* for ReplaceShip that expands the Sail action.

Variable Constraining Metaplans

In addition to the plan-refining choice of subplans and exploration of subactions, the other plan-building task is the instantiation of the free variables found in the plans. Such variables may either be directly instantiated to a specified value, as modeled by the *instantiate-var* metaplan, or more gradually constrained to subsets of the possible values, as modeled by *add-constraint*.

The *instantiate-var* metaplan reflects the agent's choice of a particular entity to instantiate an open variable in the current plan. For example, the ReplaceShip plan in Figure 2 (3) introduces a free variable for the ?new-ship. If the agent were to choose the Roark as a replacement vessel, that would be modeled by an *instantiate-var* metaplan attached to the *build-plan* node that first introduced the variable, as shown in Figure 3.

(*build-plan* (ReplaceShip Knox ?new-ship)) (1)
  (*instantiate-var* ?new-ship Roark) (2)
    (*build-plan* (ReplaceShip Knox Roark)) (3)

Figure 3: *Instantiate-Var*

The agent may also constrain the possible values for a free variable without instantiating it by using a predicate to filter the set of possible fillers. For example, the agent might decide to consider as replacement vessels only those that are within 500 miles of the damaged one. The predicate from the *add-constraint* node in line (2) of Figure 4 is inherited by the lower *build-plan* node (3), which thus represents the agent's consideration of the smaller class of plans where the value of ?new-ship satisfies the added constraint.

(*build-plan*
  (ReplaceShip Knox ?new-ship)) (1)
  (*add-constraint*
    ?new-ship
    (< (distance Knox ?new-ship) 500)) (2)
    (*build-plan*
      (ReplaceShip Knox ?new-ship)) (3)

Figure 4: *Add-Constraint*

The metaplan context tree thus inherits its basic structure from the domain plans as reflected in the *build-plan*, *build-subplan*, and *build-subaction* nodes, and as further specified by the instantiation of domain plan variables recorded in *instantiate-var* and *add-constraint* nodes. Because the domain plans occur as arguments to the plan-building metaplans, the metaplan tree turns out to include all the information that would be available from a normal domain plan context tree, so that no separate domain tree structure is needed.

3. QUERY METAPLANS

Although the plan-building metaplans that model the exploration of possible plans and the gradual refinement of an intended plan represent the agent's underlying intent, such moves are seldom observed directly in the expert advising setting. The agent's main observable actions are queries of various sorts, requests for information to guide the plan-building choices. While these queries do not directly add structure to the domain plan being considered, they do provide the expert with indirect evidence as to the plan-building choices the agent is considering. A key advantage of the metaplan approach is the precision with which it models the space of possible queries motivated by a given plan-building context, which in turn makes it easier to predict underlying plan-building structure based on the observed queries. The query metaplans include both plan feasibility queries about plan preconditions and slot data queries that ask about the possible fillers for free variables.

Plan Feasibility Queries

The simplest feasibility query metaplan is *ask-pred-value*, which models at any *build-plan* node a query for a relevant value from one of the preconditions of that domain plan. For example, recalling the original IncreaseGroupReadiness context in which the Knox had been damaged, if the agent's first query in that context is "Where is Knox?", the expert's task becomes to extend the context model in a way that explains the occurrence of that query. While that search would need to explore various paths, one match can be found by applying the sequence of metaplans shown in Figure 5.

(*build-plan*
  (IncreaseGroupReadiness
    Knox-group poor good)) (1)
  (*build-subplan*
    (IncreaseGroupReadiness ...)
    (ReplaceShip ...)) (2)
    (*build-plan*
      (ReplaceShip Knox ?new-ship)) (3)
      (*ask-pred-value*
        (ReplaceShip Knox ?new-ship)
        (location-of Knox Knox-loc)) (4)

Figure 5: *Ask-Pred-Value*

The *build-subplan* (2) and *build-plan* (3) nodes, as before, model the agent's choice to consider replacing the damaged ship. Because the ReplaceShip domain plan includes among its preconditions (not shown here) a predicate for the location of the damaged ship as the destination for the replacement, the *ask-pred-value* metaplan (4) can then match this query, explaining the agent's question as occasioned by exploration of the ReplaceShip plan. Clearly, there may in general be many metaplan derivations that can justify a given query. In this example, the RepairShip plan might also refer to the location of the damaged ship as the destination for transporting spare parts, so that this query might also arise from consideration of that plan. Use of such a model thus requires heuristic methods for maintaining and ranking alternative paths, but those are not described here.

The other type of plan feasibility query is *check-pred-value*, where the agent asks a yes/no query about the value of a precondition. As an example of that in a context that also happens to require a deeper search than the previous example, suppose the agent followed the previous query with "Is Roark in the Suez?". Figure 6 shows one branch the search would follow, building down from the *build-plan* for ReplaceShip in Figure 5 (3).

```
(build-plan
   (ReplaceShip Knox ?new-ship))                    (1)
 (instantiate-var
    (ReplaceShip Knox ?new-ship)
    ?new-ship Roark)                                (2)
  (build-plan
     (ReplaceShip Knox Roark))                      (3)
   (build-subaction
      (ReplaceShip ...) (Sail ...))                 (4)
    (build-plan
       (Sail Roark Roark-loc Knox-loc))             (5)
     (check-pred-value
        (Sail Roark Roark-loc Knox-loc)
        (location-of Roark Roark-loc))              (6)
```

Figure 6: *Instantiate-Var* and *Build-Subaction*

Here the search has to go through *instantiate-var* and *build-subaction* steps. The ReplaceShip plan has a subaction (Sail ?ship ?old-loc ?new-loc) with a precondition (location-of ?ship ?old-loc) that can match the condition tested in the query. However, if the existing *build-plan* node (1) were directly expanded by *build-subaction* to a *build-plan* for Sail, the ?new-ship variable would not be bound, so that that path would not fully explain the given query. The expert instead must deduce that the agent is considering the Roark as an instantiation for ReplaceShip's ?new-ship, with an *instantiate-var* plan (2) modeling that tentative instantiation and producing a *build-plan* for ReplaceShip (3) where the ?new-ship variable is properly instantiated so that its Sail *sub-action* (5) predicts the actual query correctly.

Slot Data Queries

While the feasibility queries ask about the values of plan preconditions, the slot data queries gather data about the possible values of a free plan variable. The most frequent of the slot data query metaplans is *ask-fillers*, which asks for a list of the items that are of the correct type and that satisfy some subset of the precondition requirements that apply to the filler of the free variable. For example, an *ask-fillers* node attached beneath the *build-plan* for ReplaceShip in Figure 6 (1) could model queries like "List the frigates." or "List the C1 frigates.", since the ?new-ship variable is required by the preconditions of ReplaceShip to be a frigate in the top readiness condition.

An *ask-fillers* query can also be applied to a context already restricted by an *add-constraint* metaplan to match a query that imposes a restriction not found in the plan preconditions. Thus the *ask-fillers* node in line (4) of Figure 7 would match the query "List the C1 frigates that are less than 500 miles from the Knox." since it is applied to a *build-plan* node that already inherits that added distance constraint.

```
(build-plan
   (ReplaceShip Knox ?new-ship))                    (1)
 (add-constraint
    ?new-ship
    (< (distance Knox ?new-ship) 500))              (2)
  (build-plan
     (ReplaceShip Knox ?new-ship))                  (3)
   (ask-fillers
      ?new-ship
      (ReplaceShip Knox ?new-ship))                 (4)
```

Figure 7: *Ask-Fillers*

Note that it is the query that indicates to the expert that the agent has decided to restrict consideration of possible fillers for the ?new-ship slot to those that are closest and thus can most quickly and cheaply replace the Knox, while the restriction in turn serves to make the query more efficient, since it reduces the number of items that must be included, leaving only those most likely to be useful.

There are three other slot data metaplans

that are closely related to *ask-fillers* in that they request information about the set of possible fillers but that do not request that the set be listed in full. The *ask-cardinality* metaplan requests only the size of such a set, as in the query "How many frigates are C1?". Such queries can be easier and quicker to answer than the parallel *ask-fillers* query while still supplying enough information to indicate which planning path is worth pursuing. The *check-cardinality* metaplan covers yes/no queries about the set size, and *ask-existence* covers the bare question whether the given set is empty or not, as in the query "Are there any C1 frigates within 500 miles of Knox?".

In addition to the slot data metaplans that directly represent requests for information, modeling slot data queries requires metaplans that modify the information to be returned from such a query in form or amount. There are three such query modifying metaplans, *limit-cardinality, sort-set-by-scalar,* and *ask-attribute-value.* The *limit-cardinality* modifier models a restriction by the agent on the number of values to be returned by an *ask-fillers* query, as in the queries "List 3 of the frigates." or "Name a C1 frigate within 500 miles of Knox.". The *sort-set-by-scalar* metaplan covers cases where the agent requests that the results be sorted based on some scalar function, either one known to be relevant from the plan preconditions or one the agent otherwise believes to be so. The function of *ask-attribute-value* is to request the display of additional information along with the values returned, for example, "List the frigates and how far they are from the Knox.".

These modification metaplans can be combined to model more complex queries. For example, *sort-set-by-scalar* and *ask-attribute-value* are combined in the query "List the C1 frigates in order of decreasing speed showing speed and distance from the Knox.". In the metaplan tree, branches with multiple modifying metaplans show their combined effects in the queries they will match. For example, Figure 8 shows the branch that matches the query "What are the 3 fastest frigates?". The *sort-set-by-scalar* metaplan in line (2) requests the sorting of the possible fillers of the ?new-ship slot on the basis of descending speed, and the *limit-cardinality* metaplan in that context then restricts the answer to the first 3 values on that sorted list.

As shown in these examples, the slot data query metaplans provide a model for some of the rich space of possible queries that the agent can use to get suggestions of possible fillers. Along with the plan feasibility metaplans, they model the structure of possible queries in their relationship to the agent's plan-refining and variable-instantiating moves. This tight modeling of that connection makes it possible to predict what queries might follow from a particular plan-building path and therefore also to track more accurately, given the queries, which plan-building paths the agent is actually considering.

```
(build-plan
   (ReplaceShip Knox ?new-ship))                    (1)
  (sort-set-by-scalar
     ?new-ship
     (speed-of ?new-ship ?speed)
     descending)                                    (2)
   (limit-cardinality ?new-ship 3)                  (3)
     (ask-fillers
        ?new-ship
        (ReplaceShip Knox ?new-ship))               (4)
```

Figure 8: *Sort-Set-by-Scalar and Limit-Cardinality*

4. COMPARISON WITH OTHER PLAN-BASED DISCOURSE MODELS

The use of plans to model the domain task level organization of discourse goes back to Grosz's (1977) use of a hierarchy of focus spaces derived from a task model to understand anaphora. Robinson (1980a, 1980b) subsequently used task model trees of goals and actions to interpret vague verb phrases. Some of the basic heuristics for plan recognition and plan tracking were formalized by Allen and Perrault (1980), who used their plan model of the agent's goals to provide information beyond the direct answer to the agent's query. Carberry (1983, 1984, 1985a, 1985b) has extended that into a plan-tracking model for use in interpreting pragmatic ill-formedness and intersentential ellipsis. The approach presented here builds on those uses of plans for task modeling, but adds a layer modeling problem-solving structure. One result is that the connection between queries and plans that is implemented in those approaches either directly in the system code or in sets of inference rules is implemented here by the query metaplans. Recently, Kautz (1985) has outlined a logical theory for plan tracking that makes use of a classification of plans based on their included actions. His work suggested the structure of plan classes based on effects and preconditions that is used here to represent the agent's partially specified plan during the problem-solving dialogue.

Domain plan models have also been used as elements within more complete discourse models. Carberry's model includes, along with the plan tree, a stack that records the discourse context and that she uses for predicting the discourse goals like *accept-question* or *express-surprise* that are appropriate in a given discourse state. Sidner (1983, 1985) has developed a theory of "plan parsing" for distinguishing which of the plans that the speaker has in mind are plans that the speaker also intends the hearer to recognize in order to produce the intended response. Grosz and Sidner (1985) together have recently outlined a three-part model for discourse context; in their terms, plan models capture part of the intentional structure of the discourse. The metaplan model presented here tries to capture more of that intentional structure than strictly domain plan models, rather than to be a complete model of discourse context.

The addition of metaplans to plan-based models owes much to the work of Wilensky (1983), who proposed a model in which metaplans, with other plans as arguments, were used to capture higher levels of organization in behavior like combining two different plans where some steps overlap. Wilensky's metaplans could be nested arbitrarily deeply, providing both a rich and extensive modeling tool. Litman (1985) applied metaplanning to model discourse structures like interruptions and clarification subdialogues using a stack of metaplan contexts. The approach taken here is similar to Litman's in using a metaplan component to enhance a plan-based discourse model, but the metaplans here are used for a different purpose, to model the particular strategies that shape problem-solving discourse. Instead of a small number of metaplans used to represent changes in focus among domain plans, we have a larger set modeling the problem-solving and query strategies by which the agent builds a domain plan.

Because this model uses its metaplans to capture different aspects of discourse structure than those modeled by Litman's, it also predicts other aspects of agent problem-solving behavior. Because it predicts which queries can be generated by considering particular plans, it can deduce the most closely related domain plan that could motivate a particular query. For instance, when the agent asked about frigates within 500 miles of Knox, the constraint on distance from Knox suggested that the agent was considering the ReplaceShip plan; a similar constraint on distance from port would suggest a RepairShip plan, looking for a ship to transport replacement parts to the damaged one. Another advantage of modeling this level of structure is that the metaplan nodes capture the stack of contexts on which follow-on queries might be based. In this example, follow-on queries might add a new constraint like "with fuel at 80% of capacity" as a child of the existing *add-constraint* node, add an alternative constraint like "within 1000 miles of Knox" as a sibling, query some other predicate within ReplaceShip, or attach even further up the tree. As pointed out below in Section 6, the metaplan structures presented here can also be extended to model alternate problem-solving strategies like *compare-plan* vs. *build-plan*, thus improving their predictive power through sensitivity to different typical patterns of agent movement within the metaplan tree. The clear representation of the problem-solving structure offered in this model also provides the right hooks for attaching heuristic weights to guide the plan tracking system to the most likely plan context match for each new input. Within problem-solving settings, a model that captures this level of discourse structure therefore strengthens an NL system's abilities to track the agent's plans and predict likely queries.

5. APPLICATIONS AND IMPLEMENTATION

This improved ability of the metaplan model to track the agent's problem-solving process and predict likely next moves could be applied in many of the same contexts in which domain plan models have been employed, including anaphora and ellipsis processing and generating cooperative responses. For example, consider the following dialogue where the cruiser Biddle has had an equipment failure:

Agent:	Which other cruisers are in the Indian Ocean?	(1)
Expert:	<Lists 6 cruisers>	(2)
Agent:	Any within 200 miles of Biddle?	(3)
Expert:	Horne and Belknap.	(4)
Agent:	Any of them at Diego Garcia?	(5)
Expert:	Yes, Dale, and there is a supply flight going out to Biddle tonight.	(6)

The agent first asks about other cruisers that may have the relevant spare parts. The expert can deduce from the query in line (3) that the agent is considering SupplySparePartByShip. The "them" in the next query in line (5) could refer either to all six cruisers or to just the two listed in (4). Because the model does not predict the Diego Garcia query as relevant to the current plan context, it is recognized after search in the

metaplan tree as due instead to a SupplyPartBy-Plane plan, with the change in plan context implying the correct resolution of the anaphora and also suggesting the addition of the helpful information in (6). The metaplan model of the pragmatic context thus enables the NL processing to be more robust and cooperative.

The Pragma system in which this metaplan model is being developed and tested makes use of the pragmatic model's predictions for suggesting corrections to ill-formed input. Given a suitable library of domain plans and an initial context, Pragma can expand its metaplan tree under heuristic control identifying nodes that match each new query in a coherent problem-solving dialogue and thereby building up a model of the agent's problem-solving behavior. A domain plan library for a subset of naval fleet operations plans and sets of examples in that domain have been built and tested. The resulting model has been used experimentally for dealing with input that is ill-formed due to a single localized error. Such queries can be represented as underspecified logical forms containing "wildcard" terms whose meaning is unknown due to the ill-formedness. By searching the metaplan tree for queries coherently related to the previous context, suggested fillers can be found for the unknown wildcards. For the roughly 20 examples worked with so far, Pragma returns between 1 and 3 suggested corrections for the ill-formed element in each sentence, found by searching for matching queries in its metaplan context model.

6. EXTENSIONS TO THE MODEL AND AREAS FOR FURTHER WORK

This effort to capture further levels of structure in order to better model and predict the agent's behavior needs to be extended both to achieve further coverage of the expert advising domain and to develop models on the same level for other discourse settings. The current model also includes simplifying assumptions about agent knowledge and cooperativity that should be relaxed.

Within the expert advising domain, further classes of metaplans are required to cover informing and evaluative behavior. While the expert can usually deduce the agent's plan-building progress from the queries, there are cases where that is not true. For example, an agent who was told that the nearest C1 frigate was the Wilson might respond "I don't want to use it.", a problem-solving move whose goal is to help the expert track the agent's planning correctly, predicting queries about other ships rather than further exploration of that branch. Informing metaplans would model such actions whose purpose is to inform the expert about the agent's goals or constraints in order to facilitate the expert's plan tracking. Evaluative metaplans would capture queries whose purpose was not just establishing plan feasibility but comparing the cost of different feasible plans. Such queries can involve factors like fuel consumption rates that are not strictly plan preconditions. The typical patterns of movement in the metaplan tree are also different for evaluation, where the agent may compare two differently-instantiated *build-plan* nodes point for point, moving back and forth repeatedly, rather than following the typical feasibility pattern of depth-first exploration. Such a comparison pattern is highly structured, even though it would appear to the current model as patternless alternation between *ask-pred-value* queries on two different plan branches. Metaplans that capture that layer of problem-solving strategy would thus significantly extend the power of the model.

Another important extension would be to work out the metaplan structure of other discourse settings. For an example closely related to expert advising, consider two people trying to work out a plan for a common goal; each one makes points in their discussion based on features of the possible plan classes, and the relationship between their statements and the plans and the strategy of their movements in the plan tree could be formalized in a similar system of metaplans.

The current model also depends on a number of simplifying assumptions about the cooperativeness and knowledge of the agent and expert that should be relaxed to increase its generality. For example, the model assumes that both the expert and the agent have complete and accurate knowledge of the plans and their preconditions. As Pollack (1986) has shown, the agent's plan knowledge should instead be formulated in terms of the individual beliefs that define what it means to have a plan, so the model can handle cases where the agent's plans are incomplete or incorrect. Such a model of the agent's beliefs could also be a major factor in the heuristics of plan tracking, identifying, for example, predicates whose value the agent does not already know which therefore are more likely to be queried. The current model should also be extended to handle multiple goals on the agent's part, examples where the expert does not know in advance the agent's top-level goal, and cases of interactions between plans.

However, no matter how powerful the pragmatic modeling approach becomes, there is a practical limitation in the problem-solving setting on the amount of data available to the expert in the agent's queries. More powerful, higher level models require that the expert have appropriately more data about the agent's goals and problem-solving state. That tradeoff explains why an advisor who is also a friend can often be much more helpful than an anonymous expert whose domain knowledge may be similar but whose knowledge of the agent's goals and state is weaker. The goal for cooperative interfaces must be a flexible level of pragmatic modeling that can take full advantage of all the available knowledge about the agent and the recognizable elements of discourse structure while still avoiding having to create high-level structures for which the data is not available.

REFERENCES

Allen, James F.; and Perrault, C. Raymond. 1980 Analyzing Intention in Utterances. *Artificial Intelligence* 15: 143-178.

Carberry, M. Sandra. 1983 Tracking User Goals in an Information-Seeking Environment. *Proceedings of the National Conference on Artificial Intelligence*. William Kaufmann: 59-63.

Carberry, M. Sandra. 1984 Understanding Pragmatically Ill-Formed Input. *Proceedings of the 10th International Conference on Computational Linguistics*. ACL: 200-206.

Carberry, M. Sandra. 1985a A Pragmatics-Based Approach to Understanding Intersentential Ellipsis. *Proceedings of the 23rd Annual Meeting of the ACL*: 188-197.

Carberry, M. Sandra. 1985b Pragmatic Modeling in Information System Interfaces. Ph.D. thesis, University of Delaware.

Grosz, Barbara J. 1977 The Representation and Use of Focus in a System for Understanding Dialogues. *Proceedings of IJCAI 77*. Morgan Kaufmann: 67-76.

Grosz, Barbara J.; and Sidner, Candace L. 1985 The Structures of Discourse Structure. Technical Report 6097, Bolt Beranek and Newman.

Kautz, Henry A. 1985 Toward a Theory of Plan Recognition. Technical Report 162, University of Rochester.

Litman, Diane J. 1985 Plan Recognition and Discourse Analysis: An Integrated Approach for Understanding Dialogues. Ph.D. thesis, University of Rochester.

Pollack, Martha E. 1986 Inferring Domain Plans in Question-Answering. Ph.D. thesis, University of Pennsylvania.

Robinson, Ann E. 1980a The Interpretation of Verb Phrases in Dialog. Technical Report 206, SRI International.

Robinson, Ann E.; Appelt, Douglas E.; Grosz, Barbara J.; Hendrix, Gary G.; and Robinson, Jane J. 1980b Interpreting Natural-Language Utterances in Dialogs About Tasks. Technical Report 210, SRI International.

Sidner, Candace L. 1983 What the Speaker Means: The Recognition of Speaker's Plans in Discourse. *International Journal of Computers and Mathematics* 9(1): 71-82.

Sidner, Candace L. 1985 Plan Parsing for Intended Response Recognition in Discourse. *Computational Intelligence* 1: 1-10.

Wilensky, Robert. 1983 *Planning and Understanding*. Addison-Wesley.