A Dialogue Model for Interaction with Planners, Schedulers and Executives

Nate Blaylock*
Dept. of Computer Science
University of Rochester
Rochester, New York, USA
blaylock@cs.rochester.edu

John Dowding
RIACS
NASA Ames Research Center
Moffett Field, California, USA
jdowding@riacs.edu

James Allen
Dept. of Computer Science
University of Rochester
Rochester, New York, USA
james@cs.rochester.edu

Abstract

We present a collaborative problem-solving model for interaction between humans and automated systems. Because the model is based on problem solving (i.e., planning and acting), it provides a natural interface between a preexisting dialogue system and preexisting planners, schedulers, and/or executives. We also describe an abstract architecture for building a dialogue system interface to planners/schedulers/executives in a highly domain-independent and portable way.

Introduction

Many planners, schedulers, and executives (henceforth *plan systems*) are built to function without human intervention. Most planners, for example, take a goal (or set of goals) as input, work on the problem and then output a plan. Most schedulers and executives work in much the same way: once the task has been specified, they go do it and only come back when they are done.

These "stand-alone" systems are desirable in some domains. The Remote Agent system (Muscettola *et al.* 1998), for example, was designed to pilot an unmanned deep space probe: an environment in which no humans are around to interact with at all! However, in domains where humans are actually present, it is not clear that it is always desirable to treat plan systems as "stand-alone" systems.

Tor example, it is probably desirable to have some outside control of the system for executives. Science fiction is filled with stories of automated systems gone awry. More practically, things such as plan failure or exogenous events may change the situation substantially enough that the user should be notified and high-level goals reevaluated.

Human interaction can also be helpful to planners. Humans are adept at looking at the overall situation and noticing heuristics or constraints that can help significantly narrow the search space for a solution. Such gestalt heuristics are not always obvious to planners. In addition, interaction can allow the user to lead the system to preferred solutions.

*Also affiliated with RIACS

User:	"What is the schedule for satellite 1 tomorrow?"
System:	"Satellite 1 is scheduled to observe sector 5 from 10:00 to 13:00 and will otherwise be idle."
User:	"Schedule it to observe sector 22 from 14:00 to 18:00."
System:	"OK."

Figure 1: Sample Dialogue of Scheduling Earth Observing Satellites (EOSs)

There are a wide range of ways to interact with plan systems, which we call *collaboration paradigms*. On one end of the spectrum is the *master-slave* paradigm (Pollack 1986), where one of the agents is completely passive and simply responds to requests from the other agent. The other end of the spectrum is *mixed-initiative* interaction, where both agents actively participate in the interaction.

Obviously, paradigms on the mixed-initiative end of the scale require a large amount of system autonomy. Also, different paradigms are probably better suited to certain domains or desirable for different applications. We do not advocate any particular paradigm here. Rather, we present a general framework in which any of these paradigms can be used for interaction with plan systems. Before we actually go on to describe this framework, we give several example dialogues (Figures 1–3) which show several types of possible interaction.

Figure 1 shows a sample dialogue of interacting with a scheduler for Earth Observing Satellites (EOSs). This shows a very basic level of interaction where the user can query about the future schedule and make adjustments.

Figure 2 shows a slightly more complex interaction level, here with an executive in a Mars Rover robot. The queries are about the plan as well as the current state of execution. The user also has the ability to control the robot's execution to avert any undesired situations — in this case to keep the robot from going off into a dust storm.

User:	"What will you be doing at 16:00?"
System:	"I will be going to the west rock formation to collect samples."
User:	"What are you doing now?"
System:	"I am going to sector 7 to take readings."
User:	"Come back to the ship. There is a dust storm in that sector."
System:	"OK, I'm coming back."

Figure 2: Sample Dialogue of Interaction with a Mars Rover Executive

User:	"We need to get some rock samples from sectors 1 through 5 today."
System:	"Rovers 1, 3 and 5 are already scheduled for today, but Rovers 2 and 4 are free."
User:	"How many samples can each rover carry?"
System:	"Two."
User:	"Let's send Rover 2 to get samples from sectors 1 and 2."
System:	"Actually, Rover 4 is closer to that area, should we send it instead?"
User:	"Yes, let's do that."

Figure 3: Sample Dialogue of Mixed-Initiative Planning with a Mars Rover Planner

Figure 3 shows a mixed-initiative exchange between a user and a central planner for several Mars Rovers. Here both the user and the system actively make suggestions and evaluate each other's suggestions. The system suggests a change in the user's recipe because it knows that Rover 4 is closer to the desired destination than Rover 2. This kind of interaction allows both parties to use their complementary skills and knowledge to construct a plan that alone, neither would be able to construct (cf. (Ferguson, Allen, & Miller 1996)).

In the remaining sections, we present a flexible framework in which to combine dialogue systems and preexisting planners, schedulers, and/or executives. The first part of this framework is a model of interaction which can handle the range of collaboration paradigms discussed above. The model is used to motivate a domain-independent communication language between the disparate components.

The second part of the framework is an abstract architecture with an interface component that connects the dialogue system to the planning component.

Although this framework does not provide the specifics of building a dialogue interface for any particular system, it does provide a uniform way to build such systems, and the interaction model provides an interface for more easily porting to new domains and components.

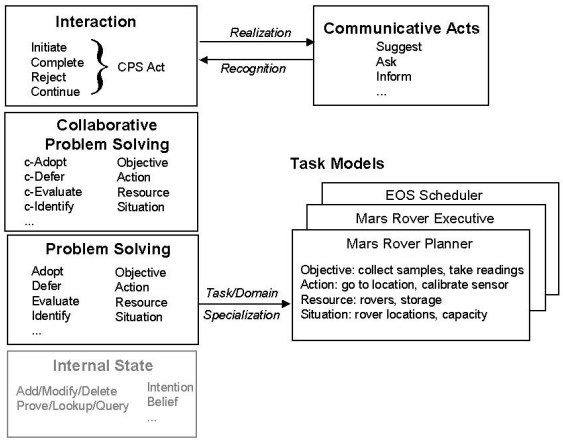

Figure 4: Collaborative Problem-Solving Model (adapted from (Allen, Blaylock, & Ferguson 2002))

Collaborative Problem-Solving Model

As shown by the examples above, spoken interaction with a planner, scheduler, or executive can involve much more than just adding or removing goals and constraints. It is true that this is a part of the interaction, and in a way, it is one of the ends of the interaction. However, a lot of other activities take place along the way.

We model human-computer interaction as collaborative problem solving (CPS) between agents, where the agents work together to accomplish their goals.[1] The central objects of this model are goals and plans (which, in order to avoid terminology conflicts, we call *objectives* and *recipes*, respectively). Problem solving can be divided into three activities:

1. Deciding on objectives (goals)
2. Deciding on recipes (plans) to accomplish the objectives
3. Using recipes to act

Each of these activities typically involves a lot of interaction. Activities can also be interleaved, as agents may be working simultaneously on several objectives, or acting (executing a recipe) in order to aid planning decisions for higher-level objectives. Agents may also revisit and change past decisions, change recipes for an objective, or even drop certain objectives and adopt others.

As an example of problem solving at a collaborative level (i.e., with more than one participant), consider again the dialogue in Figure 3. Here, the user's first utterance is used to identify and adopt an objective

[1]Space constraints allow us only to give a brief description of this model. For more details please see (Blaylock 2002; Allen, Blaylock, & Ferguson 2002; Blaylock, Allen, & Ferguson 2002).

to work on (getting the rock samples). The system then (implicitly) accepts that objective and identifies some resources (the free rovers) which can be used in achieving it. After some information about the situation is identified (i.e., rover capacity), the user suggests adopting a partial recipe (using Rover 2 to get certain samples). The system suggests changing the resource used in the recipe (Rover 4 instead) and the user agrees. At this point the conversation continues, as they have more objectives to accomplish (i.e., getting the other samples).

In collaborative problem solving, we view actions by individual agents as attempts to make changes in the joint collaborative problem-solving (CPS) *state*, or the joint commitments of the agents towards objectives, recipes, and so forth.

As shown in Figure 4, the CPS model defines a set of abstract objects of commitment which are specialized by domain-specific *task models*. It also defines a set of operators on those objects at both the single-agent problem solving and collaborative problem-solving levels. Execution of these operators causes changes in the problem solving state (either single-agent or collaborative).

The interaction level is needed because of the asymmetry between the single-agent and collaborative problem solving. Collaborating agents cannot individually change the CPS state. Doing so would imply forcing direct changes in another agent's mental state. Instead, individual agents execute *interaction acts*, which are actions used to negotiate changes in the CPS state.

Interaction acts can be used directly as an artificial agent communication language. If the modality of the interaction is natural language, however, interaction acts are realized linguistically as communicative acts.

In the remainder of this section, we describe task models, then the problem-solving, collaborative problem-solving and interaction levels. We then show an example dialogue and discuss the coverage of the model.

Task Models

Problem-solving (PS) objects are abstract objects which are instantiated by a domain-specific task model. This allows the model to interface with specific domains. The following are the PS objects:

Objectives: Assertions that are considered goals in the domain.

Recipes: Plans for achieving objectives.

Atomic Actions: Domain actions that are directly executable by the agent.

Resources: Objects which may be used in a plan.

Situations: The state of the world (or a possible world).

These abstractions allow the rest of the model to operate on the domain-independent concepts of objectives, recipes, etc. Acts at both the single-agent and collaborative levels take PS objects as arguments.

Collaborative Problem-Solving Acts

At the heart of the interaction model are collaborative problem-solving (CPS) acts, which are multi-agent actions that change the CPS state. In order to see how agents collaborate to solve problems, it is instructive to first look at an abstract model of how a single-agent solves problems, which can then be extended to the multi-agent level. We first describe at an abstract single-agent problem-solving model and then generalize it to a collaborative problem-solving model.

Single-Agent Problem Solving

The problem-solving (PS) level describes problem solving for a single agent. With this model, we are not trying to compete with formal models of agent behavior (e.g., (Cohen & Levesque 1990; Rao & Georgeff 1991)), which use agent intentions, beliefs and so forth to predict agent action and changes in the agent's state. Our PS model describes possible changes in the agent's problem-solving state at the granularity at which collaboration occurs.

As discussed above, an agent's problem solving is divided into three activities which may be freely interleaved:

1. Deciding on objectives
2. Deciding on recipes to accomplish the objectives
3. Using recipes to act

In our model, problem-solving (PS) acts are the operators which an agent uses to affect its problem-solving state. These can be classified into two groups: acts relating to commitment and acts related to reasoning.

Acts Relating to Commitment These acts change the agent's commitment to the various PS objects. The acts are listed here.[2]

- *Adopt*: Commits the agent to the PS object, i.e., the agent intends to do/use/pursue the object.
- *Select*: Moves an adopted object into the small set which is currently influencing the agent's behavior (i.e., in acting).
- *Defer*: Tables a selected object. The agent is still committed to the object, but it is no longer influencing the agent's current behavior.
- *Abandon*: Removes an agent's commitment to an object which has not yet fulfilled its purpose.
- *Release*: Removes an agent's commitment to an object which *has* fulfilled its purpose.

[2]We do not claim that this is a complete list. We do, however, believe that the final list will be short.

These are better understood in context with the PS objects to which they are applied. We describe the effect of these PS acts when applied to each of the PS objects. (Note that not all PS acts are applicable to all PS objects.)

Objectives: An agent can **adopt** an objective, which commits the agent to pursuing it. An agent may have any number of adopted objectives, but there is only a small subset which is active, controlling the agent's current activity. **Selecting** an objective makes it active, meaning that the agent is currently actively pursuing it (by executing some recipe). An agent can demote an active objective back to the adopted state by **deferring** it. An agent can also, at any time, **abandon** an objective, dropping its commitment to it. Finally, when an agent believes that an adopted objective has been attained, it may **release** it. Note that an objective is not automatically released when it is attained. An agent must *believe* that it has been attained and then consciously release it.

Recipes: These do not quite fit the same model as objectives, since it does not make sense for an agent to be committed to more than one recipe for a single objective. An agent commits to a recipe for an objective by **adopting** it. An agent can also **release** or **abandon** an adopted recipe, similar to objectives. **Select** and **defer** do not play a role with recipes.

Atomic Actions: These are similar to objectives. **Adopting** an action means that an agent is committed to performing it. A **selected** action is something that is being executed at the current time, which can be **deferred** (to suspend execution[3]), **abandoned**, or **released**.

Resources: These are objects that are somehow used in a recipe. They fit the same model as recipes in that only one resource is committed to for a certain slot in a recipe. Agents can **adopt** a resource, committing to use it in a recipe. They can also **abandon** or **release** a resource. **Select** and **defer** are not applicable to resources.

Situations: These are somewhat similar to recipes and resources. What is commitment to a situation? We believe this is used to explain "what if" type, possible worlds simulations. When an agent **adopts** a situation, it is committing to do reasoning according to that situation. **Abandoning** a situation reverts all reasoning to the *actual* situation. It is unclear what **releasing** a situation would mean. **Select** and **defer** also do not play a role with situations.

Figure 5 gives a synopsis of applicability of commitment PS acts. An `x` shows that a PS act is applicable to the object and a blank indicates non-applicability.

	Adopt	Select	Defer	Abandon	Release
Objective	X	X	X	X	X
Recipe	X			X	X
Action	X	X	X	X	X
Resource	X			X	X
Situation	X			X	

Figure 5: Application of Commitment PS Acts to PS Objects

Acts Relating to Reasoning An agent must do reasoning to decide which commitments to adopt, abandon, and so forth. This is accomplished with the following reasoning PS acts.[4]

Identify: Brings an object into focus. Used to determine which options are available.

Evaluate: Determines the goodness of an object in relations to its purpose.

Modify: Changes an object in some way.

We discuss these in the context of each PS object.

Objectives: When an agent is considering choosing, releasing, abandoning, etc. an objective, it must **identify** one or more objectives that are possibilities. It can then **evaluate** an objective to determine if it's a good idea to adopt it (or abandon it, etc.) The agent can also **modify** the objective in a certain way (make it more specific, change a parameter, etc.)

Recipes: These are treated very similarly to objectives. An agent can **identify** or **evaluate** possible recipes for a specific objective. **Modifying** a recipe is essentially planning and allows for novel recipes to be formulated by the agent as necessity arises.

Atomic Actions: The effect of reasoning PS acts on atomic actions are dependent on the context of whether the agent is planning (modifying a recipe) or acting. **Identifying** an action for planning identifies a possible action to be used in the recipe. In execution, it means that the agent queries the adopted recipe to identify the next step for execution. **Evaluating** an action in a planning context determines whether or not it is a good candidate for use in the recipe. In an execution context, an agent may choose to evaluate an action identified from the recipe before committing to actually perform the action (there is no constraint that an adopted recipe actually lead to attaining the objective). An action may be **modified** in either execution or planning.

Resources: These can be **identified** as candidates for use in a recipe. An agent can then **evaluate** the goodness of a resource in relation to a recipe. It is not clear what it would mean to **modify** a resource.

Situations: In all but the most trivial domains, an agent will not know the entire situation. **Identifying**

[3]When the action is suspendable

[4]Again, we do not claim this to be a comprehensive list, although we believe the final list will be small.

	Identify	Evaluate	Modify
Objective	X	X	X
Recipe	X	X	X
Action	X	X	X
Resource	X	X	
Situation	X	X	X

Figure 6: Application of Reasoning PS Acts to PS Objects

a situation gives the agent more information about what the state of that world is. An agent **evaluates** a situation in order to decide if it is a desirable situation to be in (this may also be helpful for an agent to decide whether or not an objective can be released). Hypothetical situations can be **modified** as well.

Figure 6 gives a summary of the applicability of reasoning PS acts to PS objects. Similar to figure 5, an `x` shows the applicability of acts to objects.

The Collaborative Level Collaborative problem solving is an extension of single-agent problem solving. The granularity of acts that we discussed above is the granularity at which collaboration occurs between agents. In fact, these acts are more overt in collaborative problem solving since agents must communicate and coordinate their reasoning and commitments.

At the CPS level we have CPS acts which apply to PS objects, paralleling the single-agent PS model. In order to distinguish acts at the two levels, we append a *c-* before CPS acts, creating *c-adopt*, *c-select*, *c-modify*, and so forth. CPS acts have similar intuitive meaning to those at the PS level and we will not redefine them here.[5]

Interaction Acts

An individual agent cannot perform a CPS act alone, as doing such would imply directly changing the internal state of another agent. Rather, CPS acts are generated by the individual *interaction acts* of each agent. An interaction act is a single-agent action and takes a CPS act as an argument. The interaction acts are *initiate*, *continue*, *complete* and *reject*. These are defined by their effects and are similar to the grounding model proposed in (Traum 1994).

An agent beginning a new proposal performs an *initiate*. In the case of successful generation of the CPS act, the proposal is possibly passed back and forth between the agents, being revised with *continues*, until both agents finally agree on it, which is signified by the final agent in the chain not adding any new information to the proposal but simply accepting it with a *complete*; this generates the proposed CPS act. Of course, at any point in this exchange, either agent can perform a *reject*, which causes the proposed CPS act to fail. This ability of either agent to negotiate and/or reject proposals allows our model to handle not just the master-slave collaboration paradigm, but the whole range of collaboration paradigms (including mixed-initiative) which we discussed above.

[5]Joint objects also entail slightly different individual intentions as described in (Grosz & Kraus 1996).

Example

To illustrate the model, we apply it to the sample dialogue given in Figure 3. It is reprinted and annotated with its corresponding interaction acts in Figure 7.

The first utterance is initiating an adopt of an objective[6], or suggesting that the objective be taken on. In the second utterance, the system (implicitly) completes this CPS act. This generates the CPS act *c-adopt objective* which causes a change in the CPS state. In addition, the second utterance also initiates identifying resources which can be used in attaining the objective.

The user completes this act and does an identify situation to find out more about the state of the world. The system response adds information to the CPS act (namely the answer), so it is a continue of it, which the user subsequently completes (or accepts).

In the rest of the dialogue, the user suggests a recipe to use and a resource (the rover) for the recipe. The system rejects that resource and suggests another, which the user accepts.

Coverage of the Model

This model is intended to describe most possible human-computer interaction, modeling it as collaborative problem solving. More narrowly, we believe it can be used to described most interactions between a human user and a planner, scheduler, and/or executive. The true test of coverage will likely need to be pragmatic, as we are not aware of another taxonomy of collaborative problem-solving interaction against which coverage could be measured. As we discuss in the future work section, we are currently implementing systems in several diverse domains as well as annotating corpora in even more domains as an attempt to judge the adequacy of the model as well as expand it where necessary. We briefly describe here some phenomena which can be handled by the model as well as some that are not presently handled.

Some phenomena which the model covers include: goal selection (with *adopt objective*), (interleaved) planning and execution, plan queries and summary (with *identify recipe*, *action*, etc.), plan and goal evaluations, negotiation and mixed-initiative interaction (with interaction acts), and possible worlds reasoning and planning (with *situations*). Again, we do not claim to have the actual reasoning components which would produce these activities, rather we have a model which can describe the interaction involving them.

[6]Because of space limitations we have omitted several acts in this example. For example, this first utterance is also an `(initiate (c-identify objective))`.

User: **"We need to get some rock samples from sectors 1 through 5 today."**
(initiate (c-adopt objective))
System: **"Rovers 1, 3 and 5 are already scheduled for today, but Rovers 2 and 4 are free."**
(complete (c-adopt objective)) — (initiate (c-identify resource))
User: **"How many samples can each rover carry?"**
(complete (c-identify resource)) — (initiate (c-identify situation))
System: **"Two."**
(continue (c-identify situation))
User: **"Let's send Rover 2 to get samples from sectors 1 and 2."**
(complete (c-identify situation)) — (initiate (c-adopt recipe)) — (initiate (c-adopt resource))
System: **"Actually, Rover 4 is closer to that area, should we plan on sending it instead?"**
(complete (c-adopt recipe)) — (reject (c-adopt resource)) — (initiate (c-adopt resource))
User: **"Yes, let's do that."**
(complete (c-adopt resource))

Figure 7: Sample Dialogue with Corresponding Interaction Acts

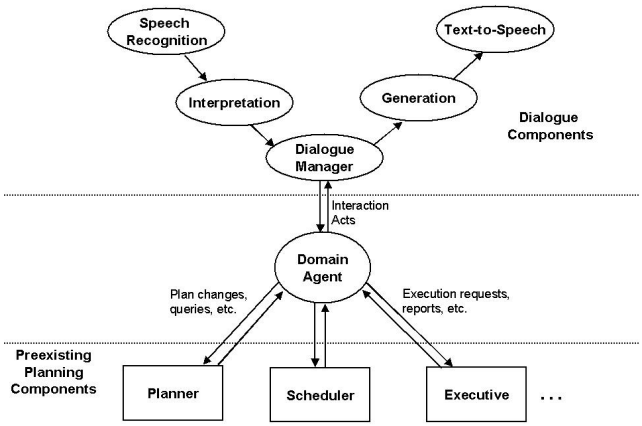

Figure 8: Architecture for System Integration

Phenomena not yet directly included in the model include: team formation — where agents come together and negotiate to start the collaboration (cf. (Wooldridge & Jennings 1999)) and global constraint and preference setting — such as "Always prefer the fastest solution" (cf. (Lemon *et al.* 2002)).

System Architecture

The CPS model gives us a sufficiently descriptive model to allow a variety of potential interactions with agents of varying abilities. The model is important in that it gives a non-linguistic, domain-independent description of collaboration which we can utilize for communication between the dialogue front end and the system with which we are interfacing.

We have still not discussed, however, how one would go about actually building a dialogue interface to a planner/scheduler/executive. In this section, we describe a generic architecture for building a dialogue system for a preexisting planning component by introducing a new agent into the architecture: the *domain agent*.

Figure 8 shows the abstract architecture. The top portion of the figure shows a simplified view of a traditional dialogue front end. An utterance from the user is sent to speech recognition after which it is interpreted to give some sort of semantic form. The dialogue manager further interprets the utterance in the context of the dialogue. It then decides what action to take based on the utterance, invokes back-end agents and then formulates a response, which is sent to generation and ultimately text-to-speech.

In most dialogue systems, the dialogue manager has to be specifically designed to interact with particular back-end components with which it communicates in an ad-hoc fashion. One of the purposes of our CPS model is to motivate a principled, communication language for the dialogue manager and back-end components. In this way, the dialogue manager can remain independent of the back-end components and back-end components are insulated from the linguistic features and only need to be able to understand collaborative problem solving. This is useful, since having to redesign dialogue managers for new domains is a serious issue in system portability.

Of course, most preexisting planners, schedulers, and executives were not designed to interface directly using interaction acts. The architecture uses an intermediary agent, known as the domain agent, to provide an interface that translates interaction acts into commands to be sent to the preexisting back-end components. The domain agent can be as simple as a wrapper for the component or as complex as an autonomous agent that gives mixed-initiative behavior to the system and does complex plan reasoning. This is where the system designer can decide what type of collaboration paradigm is desired for the application and what types of interaction should be supported.

While the idea of a "domain agent" is not new (cf. (Allen, Ferguson, & Stent 2001; Allen & Fergu-

son 2002))[7], there are several benefits from combining this with the CPS model. First, using interaction acts to motivate a communication language allows the dialogue manager to be independent of back-end components. It only needs to translate linguistic forms into interaction acts (for interpretation) and interaction acts to linguistic forms (for generation).

Perhaps more importantly is that, since the CPS model is based on joint plans and execution between the user and system, it provides a natural interface to actual planners, schedulers and executives, making it much simpler to build domain agents for preexisting systems.

Currently, we are working on integrating a preexisting dialogue system with a preexisting planner/scheduler and executive (Dowding *et al.* 2002) based on this architecture.

Related Work

Most dialogue systems (e.g., (Lamel *et al.* 1998; Rudnicky *et al.* 1999)) do dialogue management by recognizing arguments that fill in values in predefined frames. These are more apt for domains where the plan is already constructed and the user is only choosing parameters for that plan (such as travel reservations). It is unclear how such architectures could be applied to the more complex task of interacting with plan systems to create and execute plans. Although several dialogue systems (Ramshaw 1991; Chu-Carroll & Carberry 2000) do reason directly about planning, they do not handle both planning and execution.

While much work has been done on formalizing joint plans and activity (Searle 1990; Levesque, Cohen, & Nunes 1990; Grosz & Kraus 1996), relatively little has looked at the collaboration that takes place between agents that are jointly planning and/or acting.

One line of research that models collaboration is the SharedPlan formalism (Grosz & Kraus 1996) (and the derivative COLLAGEN project (Rich, Sidner, & Lesh 2001)) in which agents collaborate together to form a joint plan using four operators: *Select_Rec* — an individual agent selects a recipe to be used to attain a given subgoal; *Elaborate_Individual* — an individual agent decomposes a recipe into (eventually) completely specified atomic actions; *Select_Rec_GR* — intuitively, the same as Select_Rec, only at the multi-agent level,[8] i.e., a group of agents select a recipe for a subgoal; and *Elaborate_Group* — the multi-agent equivalent of Elaborate_Individual, i.e., a group of agents decompose a recipe.

The main focus of the SharedPlan model has been to formalize agent intentions and beliefs in forming and sharing joint plans, something which is still weak in our model. However, for our purposes, building an interface to plan systems, there are several shortcomings in this model.

First, SharedPlans only models collaboration for joint *planning* between agents. It does not model the collaboration that occurs when agents are trying to *execute* a joint plan (although it does specify the needed intentions and beliefs for agents executing joint plans). In the domains we are working in, the user and system may collaborate in both planning and acting. This sometimes happens in a serial fashion (the agents formulate a joint plan and then execute it), or it can be interleaved (the agents begin to execute a partial plan and plan "as they go"). We need to be able to model collaboration involving (possibly interleaved) planning and acting.

Second, the SharedPlans formalism models the formulation of joint plans with the four operators previously discussed: *Select_Rec*, *Elaborate_Individual*, *Select_Rec_GR*, and *Elaborate_Group*. Although these operators were sufficient to allow the formalization of group intentions and beliefs about joint plans, they do not provide enough detail for us to model collaboration at an utterance-by-utterance level (which is needed to represent communicative intentions). As an example, consider the *Elaborate_Group* operator, which has the function of decomposing a recipe, instantiating the parameters (including which agent or subgroup will perform which action at what time and which resources will be used), and making sure the rest of the group has similar intentions and beliefs about the plan. An *Elaborate_Group* can and often does consist of many individual utterances. In order to build a dialogue system, we need to be able to model the communicative intentions behind a single utterance.

We believe that our model may be compatible with SharedPlans and can be seen as specifying the details of the SharedPlan operators at an utterance level.

Future Work

We are developing several systems which utilize the CPS model in several domains including medication scheduling (Ferguson *et al.* 2002), emergency and rescue operations (Allen *et al.* 2000), and an interface to the Personal Satellite Assistant (PSA), a robot under development to be deployed on the International Space Station (Dowding *et al.* 2002). We are also annotating dialogues in a range of domains (both planning and scheduling and others) to ensure that the CPS model covers a wide range of interaction and to ensure that it is truly domain independent.

Also, as mentioned above, we are currently expanding the PSA system described in (Dowding *et al.* 2002) to interface to a planner and executive based on the dialogue architecture discussed above. We also plan to extend the system to perform more plan and task

[7] In terms of architecture in the TRIPS dialogue system, the domain agent encompasses roughly the Behavioral Agent and the back-end interaction of the Task Manager.

[8] Individual and group operators entail different constraints on individual intentions and beliefs. However, this is not important for understanding the formalism as a model of collaborative planning.

reasoning to support more mixed-initiative interaction with the user.

Conclusion

We have presented a domain-independent interaction model for planners, schedulers and executives based on collaborative problem solving. This model supports a wide range of collaboration paradigms and allows a system designer flexibility in implementing the desired type of collaboration.

In addition, this interaction model is used to motivate a domain-independent communication language between a dialogue front end and planning back-end. We presented a generic architecture which allows pre-existing plan systems to be more easily integrated with a dialogue system.

Acknowledgments

We would like to thank Jeremy Frank, Susana Early, and the anonymous reviewers for many helpful comments on this paper.

This material is based upon work supported (in part) by Department of Education (GAANN) grant no. P200A000306; ONR research grant no. N00014-01-1-1015; DARPA research grant no. F30602-98-2-0133; NSF grant no. EIA-0080124; a grant from the W. M. Keck Foundation; and support from RIACS.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above-mentioned organizations.

References

Allen, J., and Ferguson, G. 2002. Human-machine collaborative planning. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Allen, J.; Byron, D.; Dzikovska, M.; Ferguson, G.; Galescu, L.; and Stent, A. 2000. An architecture for a generic dialogue shell. *Journal of Natural Language Engineering special issue on Best Practices in Spoken Language Dialogue Systems Engineering* 6(3):1–16.

Allen, J.; Blaylock, N.; and Ferguson, G. 2002. A problem-solving model for collaborative agents. In *First International Joint Conference on Autonomous Agents and Multiagent Systems*.

Allen, J.; Ferguson, G.; and Stent, A. 2001. An architecture for more realistic conversational systems. In *Proceedings of Intelligent User Interfaces 2001 (IUI-01)*, 1–8.

Blaylock, N.; Allen, J.; and Ferguson, G. 2002. Managing communicative intentions with collaborative problem solving. In Smith, R., and van Kuppevelt, J., eds., *Current and New Directions in Discourse and Dialogue*. Kluwer. To appear.

Blaylock, N. 2002. Managing communicative intentions in dialogue using a collaborative problem-solving model. Technical Report 774, University of Rochester, Department of Computer Science.

Chu-Carroll, J., and Carberry, S. 2000. Conflict resolution in collaborative planning dialogues. *International Journal of Human-Computer Studies* 53(6):969–1015.

Cohen, P. R., and Levesque, H. J. 1990. Intention is choice with commitment. *Artificial Intelligence* 42:213–261.

Dowding, J.; Frank, J.; Hockey, B. A.; Jónsson, A.; Aist, G.; and Hieronymus, J. 2002. A spoken dialogue interface to the EUROPA planner. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Ferguson, G.; Allen, J.; and Miller, B. 1996. TRAINS-95: Towards a mixed-initiative planning assistant. In Drabble, B., ed., *The Third International Conference on Artificial Intelligence Planning Systems*, 70–77. Edinburgh, Scotland: AAAI Press.

Ferguson, G.; Allen, J.; Blaylock, N.; Byron, D.; Chambers, N.; Dzikovska, M.; Galescu, L.; Shen, X.; Swier, R.; and Swift, M. 2002. The Medication Advisor project: Preliminary report. Technical Report 776, University of Rochester, Department of Computer Science.

Grosz, B. J., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.

Lamel, L.; Rosset, S.; Gauvain, J. L.; Bennacef, S.; Garnier-Rizet, M.; and Prouts, B. 1998. The LIMSI ARISE system. In *Proceedings of the 4th IEEE Workshop on Interactive Voice Technology for Telecommunications Applications*, 209–214.

Lemon, O.; Gruenstein, A.; Hiatt, L.; Gullett, R.; Bratt, E.; and Peters, S. 2002. A multi-threaded dialogue system for task planning and execution. In Bos, J.; Foster, M. E.; and Matheson, C., eds., *Proceedings of the Sixth Workshop on the Semantics and Pragmatics of Dialogue*.

Levesque, H.; Cohen, P.; and Nunes, J. 1990. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 94–99. Boston: AAAI.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote Agent: To boldly go where no AI has gone before. *Artificial Intelligence* 103(1–2):5–48.

Pollack, M. 1986. Inferring domain plans in question-answering. Technical Report MS-CIS-86-40 LINC LAB 14, University of Pennsylvania. PhD thesis.

Ramshaw, L. A. 1991. A three-level model for plan exploration. In *Proceedings of the 29th ACL*, 39–46.

Rao, A. S., and Georgeff, M. P. 1991. Modeling rational agents within a BDI-architecture. In Allen, J.; Fikes, R.; and Sandewall, E., eds., *Principles of Knowledge Representation and Reasoning*, 473–484. Cambridge, Massachusetts: Morgan Kaufmann.

Rich, C.; Sidner, C. L.; and Lesh, N. 2001. COLLAGEN: Applying collaborative discourse theory to human-computer interaction. *AI Magazine* 22(4):15–25. Also available as MERL Tech Report TR-2000-38.

Rudnicky, A. I.; Thayer, E.; Constantinides, P.; Tchou, C.; Shern, R.; Lenzo, K.; Xu, W.; and Oh, A. 1999. Creating natural dialogs in the Carnegie Mellon Communicator system. In *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech-99)*, 1531–1534.

Searle, J. R. 1990. Collective intentions and actions. In Cohen, P. R.; Morgan, J.; and Pollack, M., eds., *Intentions in Communication*. Cambridge, MA: MIT Press. 401–415.

Traum, D. R. 1994. A computational theory of grounding in natural language conversation. Technical Report 545, University of Rochester, Department of Computer Science. PhD Thesis.

Wooldridge, M., and Jennings, N. R. 1999. The cooperative problem-solving process. *Journal of Logic and Computation* 9(4):563–592.