Abstract Task Specifications for Conversation Policies

Renée Elio
Department of Computing Science
University of Alberta
Alberta, CANADA T6G 2H1
(1-780) 492-9643
ree@cs.ualberta.ca

Afsaneh Haddadi
DaimlerChrylser Research
Alt-Moabit 96A,
10559 Berlin, Germany
(+49-30) 399 82 201
afsaneh.haddadi@daimlerchrysler.com

Ajit Singh
Department of Computing Science
University of Alberta
Alberta, CANADA T6G 2H1
(1-780) 492-9643
ajit@cs.ualberta.ca

ABSTRACT

Under some views, a crucial function for conversation policies is to "constrain the messages that appear on the wire,'' for there can be a many-to-many mapping between an agent's intention and the message primitive used to express that intention. In this paper, we argue that the way to constrain messages is to constrain intentions. We propose a pragmatic approach to doing this through an abstract task specification or model. Abstract task specifications are based on a simple state-space representation for problem formulation, and this representation can be used to delimit the agents' task intentions and discourse intentions. This analysis supports a flexible and pragmatic way of handling "unexpected'' messages by reference to the abstract task specification. We see an abstract task specification as one component of a publicly posted conversation policy. A simple search-assistant agent was implemented within a BDI architecture to illustrate application of these ideas.

Keywords

communication protocols, human-agent communication, tasks

1. INTRODUCTION

Cooperation between agents denotes a kind of interaction required when each of the agents has some, but not all, the information and abilities required to accomplish a task. This requires specifying the semantics and pragmatics of a conversation a sequence of messages that enable two agents to bring a task to completion. Most definitions of a conversation implicitly or explicitly appeal to the notion of "task accomplishment" or "goal achievement," although what constitutes a task or goal is interpreted quite broadly. Recently, there has been considerable interest in specifying *conversation policies* [8], which speak to a range of matters in managing lengthy conversations, from turn-taking and message time-out conventions to responding to dynamic constraints imposed by the environment [19]. Our concern here is what some researchers [9] claim is a crucial function of a broadly-defined conversation policy, which is: constraining "the messages that appear on the wire." It is argued that this need arises from the many-to-many mapping between an intention an agent might have and the specific agent communication language (ACL) primitive used to convey that intention. Indeed, a difficult matter long recognized within the discourse processing research community is the inferential machinery and theoretical underpinnings needed to associate a speaker's intention with a speaker's utterance. So it is not surprising that these same problems eventually come into play even in the realm of limited dialogues between software agents or between software agents and humans.

The call for conversation policies stems from a belief that the solution to these matters will not be found at the level of individual message primitives or performatives within an agent communcation language, such as KQML or FIPA's ACL [14,15,7]. However well-specified the semantics for a performative might be, they are under constrained with respect to the full illocutionary force of the communicative act. For example, an inform ought sometimes to be interpreted as a "suggestion" but in another context, as a "command." This in turn has given rise to a more *protocol* oriented view of ACL semantics, i.e., the specification of semantics for conversational sub-units as ways of structuring multi-message sequences between two agents [7,13, 20, 26]. This approach builds on the notion of representing dialogues and conversations among humans or software agents as state transition diagrams, a perspective which dates back at least to Winograd and Flores [27]. Under this view, dialogues are automatons or state-transition networks, just like any other program, except there is more than one actor or agent that is executing the network. In general, the use of finite-state machines to define protocols is a fair way of specifying the temporal and contextual conditions that determine what message types may or must follow another. The content of a particular message primitive, under this view, is determined in part by the protocol context in which it was delivered.

The continuing dilemma over identifying semantics for both primitive message types and message protocols is motivated, of course, by the need to have a clear, unambiguous message exchange. If we can cram *all* the nuances and distinctions into different primitive messages and protocols, then perhaps any run-time recognition and handling of intentions can be avoided. But we see several limitations to putting *all* hopes at these two levels alone. First, the run-time recognition and handling of intention seems essential for human-agent cooperation and communication. For while we may design a software agent that follows some particular communication protocol, we cannot assume that the human knows that protocol or would be inclined to abide by it, at

least in cases where user messages cannot be directly constrained (as in, say, via menu choices on a graphical interface). This makes the problem of understanding and structuring even limited conversations for cooperation more complex. Second, elevating the level of analysis from the individual performative to protocols (which we think is a crucial step) only moves the set of problems back a level. As several people have noted [e.g. 9], there is no consensus here either on what the primitive protocols are, let alone their semantics. Although this matter is in principle resolvable, protocols can only maintain what we call *local coherence* some unity between very short sequences of messages. In general, a protocol at best specifies all the possible courses of dialogue (alternative sequences of messages) that we, as designers, can predict as being possible and sufficient with respect to a specific goal or a task. Having this protocol, at each state, both agents know the possible next states and so any message that is exchanged brings the conversation to an "expected" state. For dialogue-extensive applications of the kind considered here, it is impractical, if not impossible, to develop one such protocol, as it is not possible to predict all the possible courses of dialogue that could occur in the conversation. While humans are unaware of existence of any protocols and therefore cannot be expected to abide by them anyway, in mixed-initiative dialogues among cooperating agents, we should also allow for agents to be able to shift focus from one sub-task to another as their situation demands, make queries on the side before advancing further with the task, inform each other of unexpected situations and so on. In other words, in mixed-initiative dialogues, in a given state *s*, the set of next possible (permitted) states is much larger than the set of next expected states. Hence in mixed-initiative dialogues protocols can only serve us in specifying in general 2-3 message sequences or sub-dialogues related to a sub-task in solving the larger task. In this respect, protocols ensure local coherence only.

When a dialogue expands beyond 2-3 message sequences, there must be some way to ensure *global coherence* to the entire conversation, i.e., a coherence to the way in which very short message sequences are, crudely put, patched together. And this leads to what we see as the fourth matter. Focusing on protocols alone will not fully address the primary function of a conversation policy as motivated in [9]: to *constrain the messages that are sent.* While protocol definitions do this locally, they do not do this globally. After one protocol completes, what constrains the *next* protocol? And in what sense does the concatenation of any sequence of protocols constitute a globally-coherent conversation?

The appeal to global coherence as a feature of a conversation is implied by Grice's [10] maxim of relation, which states that speakers aim to make their contributions relevant to the ongoing conversation. In other words, each speaker's contribution to the conversation relates to the utterances that come before and follow it, so that the whole conversation is *about something* [21]. Under our view, that "something" is what we call an *abstract task model*. While we fully believe that precise semantics are crucial for individual performatives and protocols, the full illocutionary force of a message sequence will be under constrained without some appeal to an abstract task specification. Simply put, the only way to constrain messages is to constrain and delimit intentions. For us, an abstract task is something like "scheduling," "negotiation", "database search", or "diagnosis." Similar notions of generic tasks and task models had been developed to support domain-independent methodologies and architectures for developing knowledge-based problem-solving systems [3]. We think that it is reasonable to assume that two agents come to a cooperative venture knowing that their (abstract) task is one of search, negotiation, diagnosis, or whatever. Regardless of what the actual domain content and domain ontology is, two cooperating agents must share an ontology for the abstract task they are jointly solving, and this ontology is different from the ontology for the actual domain.

We adopt a pragmatic approach to specifying an abstract task specification that begins with a problem formulation using a traditional state-space representation. This representation defines and delimits a set of task intentions, which in turn defines and delimits discourse intentions. Discourse intentions are advanced by discourse protocols standards for message sequences. The content of the individual peformatives that comprise the protocols is also specified by the abstract task specification. The resulting analysis supports a flexible and pragmatic handling of "unexpected" messages. In this respect, while a message may be unexpected in the context of some protocol, it cannot be undefined in the context of the abstract task specification that is jointly held by the two agents. And finally, the abstract task model allows for a very precise definition of agent roles in terms of their functionality in supplying or needing to know elements of the state-space representation for the problem. Some of the nuances associated with request messages as really being commands or suggests can be derived in part from these agent roles as given in the task specification.

As one component of a publicly posted conversation policy, an abstract task model addresses two elements of a broadly-defined conversation policy [9]: specific goal achievement policies and conversation management policies. We illustrate the features of this approach by showing its application to a simple and well-understood agent-assistant task. An agent-assistant based on this model was implemented in a beliefs-desires-intentions (BDI) architecture and we have few words to say about this implementation as well.

2. SUCCESSIVE REFINEMENT SEARCH

In this section, we briefly outline aspects of an application domain that led us to the perspectives described above. The problem called for a system agent, which we here call the DB agent (for database assistant), that would assist a human (e.g., a driver in an automobile) in a variety of tasks, which included location and retrieval of information from various databases. Generally speaking, the DB agent must identify one or more database entities that satisfy a set of constraints specified by the user. The database entities are defined by feature-value tuples. Using a database of restaurants as an example, a particular database entry corresponds to a collection of feature-value specifications like *:name Rangoon :location Palo Alto :ethnic-type Burmese :cost moderate etc*. We are interested only in cases for which the database is so large and the information request is so ill-defined, that a single, simple answer cannot be supplied. We also assume that the user may not necessarily have a definitive set of constraints in mind when starting the task. It may be more accurate therefore, to characterize the process as one of exploration. This was the motivation for viewing the DB agent and the user (or possibly a user-agent that serves as the user's proxy and knows something about the user's preferences and goals) cooperating in a successive refinement search, where the goal is to identify one or more database entries that satisfy a

particular set (or possibly different sets) of constraints. The following example dialogue between the DB agent and the user agent (UA), with some limited English gloss, illustrates various features of the lengthy communication we aimed to handle:

DB-1: What type of domain do you wish to search? Lumber stores? Pharmacies?
UA-2: I want an inexpensive food in San Francisco.
DB-3: O.K..
DB-4: There are several hundred inexpensive restaurants in San Francisco.
DB-5: What type of food do you want?
UA-6: What do you mean?
DB-7: I mean ethnic type such as Hungarian or French.
UA-8: Italian.
DB-9: O.K.
DB-10: Do you want take-out or sit-down?
UA-11: What about places on the Peninsula?
DB-12: OK, we'll leave San Francisco for the moment.

DB-13: There are 135 Italian restaurants on the Peninsula.
DB-14: A particular place or city will rapidly narrow down this set most rapidly.

UA-15: Redwood City or Menlo Park.

There are many user interface systems and applications that support some or all elements of this interaction. Putting aside matters of speech input and processing, we use a few features of this task to motivate key aspects of an abstract task model specification for a conversation policy, particularly when communication is limited to simple message exchange (and not, say, enabled via a visual interface, as in [22]). First, the two agents here have distinctly different functionalities. The DB agent is not merely relaying information about what it knows, but it is performing system computations that advance the task in a particular way. Only the system is able to perform these computations and have direct access to the results of the computations. The human agent can neither do these computations nor have direct access to their results except via the message exchange. Similarly, only the user (agent) has specific information that could determine task process or declare task completion. As such, while only the DB agent can actually compute the next state that moves the user towards the goal state, it is only the user who can decide if a goal state has been reached. We grant the DB agent some autonomy, insofar as it has an internal representation of the problem itself. It can take initiative in querying the user for required information, in learning user preferences over time, and in making suggestions about how to accomplish the apparent goal most effectively (e.g., compute the current search set's most discriminating feature that would lead to the largest reduction in the remaining entities). The important point is that what information can and must be exchanged between the agents what we below will call discourse intentions is dictated by the asymmetry of these functional roles. Additionally, the protocols followed for this message exchange whether a performative ought to be interpreted as a command or a suggestion are also constrained by these functional roles. We elaborate more on these points below.

Second, standard query sub-dialogues occur in several places. Exactly how many sub-dialogues might occur, and what their content might be, goes unspecified within any message protocol, and several such information exchanges might ensue before any advancement in the task itself is made. Third, either agent can take the initiative in advancing the task in a new direction. Therefore, the DB agent must respond effectively to these "unexpected communication acts. For example, in message UA-11, the user does not provide an answer to the question posed in message DB-10, and instead shifts the direction of the search task. Again, such a message is unexpected locally, but ought not to be unexpected within the task specifications.

3. ABSTRACT TASK SPECIFICATION

Our concern here is the analysis of the task at some level of abstraction and generality that support the specification of pragmatic abstract task model that can be part of conversation policy, just in the case where the human-to-agent and agent-to-agent communication bandwidth is limited to message exchange. Our pragmatic specification of task semantics flow directly from the traditional view of a problem-solving as movement through a state space, where each state has a direct or indirect correspondence to a world state. Such a formulation requires (a) a goal test that indicates whether a state (or path to a state) constitutes a problem solution, (b) a specification of an initial state, and (c) the specification of operators as functions performed on one state to produce a successor state. Task actions or operators are realized as inspectable preconditions that must match features in state *s* and inspectable post-conditions that define the transformation of state *s* into some successor state.

From this perspective, an abstract task model for successive refinement search may be formulated as follows. Each task state consists minimally of specifications or patterns associated with (a) a *domain* that corresponds to a currently loaded database, (b) a set of currently-satisfied *constraints* specified as feature value tuples, (c) a *search set* corresponding to the database entities that fit those constraints. Each of these objects itself has properties, e.g., the search set has a particular size and constituent members. The task operators that transform one state into another are setting and *loading* a domain database, *contracting* a search, and *expanding* a search set. The contract operator applies a newly-formulated constraint to the search set in state *s* to yield a new state with a new search set. The expand operator produces a new search set from a previous one, by relaxing a constraint in some particular way. Each task state is also explicitly characterized by how it relates to its predecessor. For example, we distinguish (for purposes of discourse intentions) between contract operations which produce search sets that are a proper subset of a previous search set from contract operations which do not.

Generally speaking, these operators are the *only* actions within this simple task domain. As such, they define the complete set of task actions about which either agent can be committed to take and hence, they correspond to the complete set of *task intentions*. Contextual features of the current task state impose some partial order on the next necessary or plausible intention to have, which in turn implicate some particular task operator. But the crucial point is that task intentions are defined by task operators that are executed on task states. As soon as you have this formulation, you have the task intentions.

The abstract task specification also serves to define *discourse intentions*. The commitment to a task intention does not entail being able to satisfy it via the execution of a appropriate task operator. Some aspect of the associate task operator's preconditions may not be satisfied. Thus, any task operator's

precondition is a possible *object of discourse* about which either agent could form a discourse intention the commitment to perform a communication act. Objects of discourse are topics that *may* or *must be* talked about, depending on each agent's role and in which agent knows what. The task specification, by reference to agent functionality, indicates which agent can bring about the satisfaction of a particular precondition. The agent that does not possess information it needs to execute its (intended) task operator must communicate with the agent that does have that information, as per the task model. Conversely, any agent with such information can (or must) offer the information to the agent which needs it. In our example search task, objects of discourse defined by task preconditions include the domain, a feature for a constraint, or a value for a feature that has already been specified for a constraint. Queries about these objects of discourse, as well as information exchanges about their required or desired values, are allowed.

A second set of *possible* discourse intentions are defined by a task operators post-conditions that define a new successor state. They are *necessary* objects of discourse if the agent without direct access to them nevertheless needs to know about them, in order to fulfill its role. This is the case in our search task the user has no access to the computation results (post-conditions of task operations), so they must be shared by the DB agent. That is, the DB agent must form discourse intentions to communicate these post conditions. But there are different tasks that would have different specifications indicating that certain post conditions of task-operators are *not to* be shared. Many agent applications involve what we might call "private" computations. For example, an agent may solicit bids as part of some contract-assignment task, and in doing so, is moving through its own internal task space, possibly determining which agents have responded or what bids are high or low. It needs certain information from bidding agents to make those transitions in its task space, but many aspects of those task state transitions are private and can be tagged as such in the abstract task specification. Usually, it is assumed in such applications that there is no need to represent that such computations are private. But if that specification were part of the abstract task model guiding the conversation, even bidding agents who adhere to the task specification can explicitly "know" that their discourse intentions must exclude queries about these private results.

In sum, we believe that once you have a state-space formulation for a problem and a specification of agent functionality, you can derive in a pragmatic but principled way a delimited set of task intentions as well as the necessary, possible, or forbidden discourse intentions. We are now at the point of having constrained messages from a top-down perspective. We turn next to message protocols.

4. PROTOCOLS AS OPERATORS

At this point, we have outlined at a general level how an abstract task model based on a state-space formulation of a problem and the relative functionalities of each agent in realizing movement through that space defines and delimits the necessary and possible task intentions and discourse intentions an agent may have. A task intention concerns advancement of a task-related goal or sub-goal and an discourse intention concerns advancement of information exchange, in support of a task intention. We believe it is important to regard and model the task intentions and intentions and discourse intentions as characterizing two distinct state spaces.

Many transitions may take place in discourse space before a single transition occurs in task space (most modeling of agent beliefs and intentions take place, under this view, in discourse space). Similarly, many transitions may occur in task space before it is appropriate or necessary to formulate a discourse intention. Because "discourse" itself is often modeled or viewed as a task with goals, we think that these distinctions between advancing the joint task and advancing joint knowledge about the task become blurry. But keeping these matters straight can greatly aid in understanding and specifying the local and extended perlocutionary effects of a structured message sequence. In this section, we elaborate on this view. The interplay of task space and discourse space is summarized in Figure 1. Whenever an agent cannot proceed in the task space, it adopts a discourse intention which causes the transition of control to discourse space until sufficient information has been gathered from the other agent to proceed in the task space (e.g., schematically illustrated as transitions ABC in Figure 1). Similarly, whenever an agent cannot proceed further in the discourse space, a task intention is formed to compute the necessary information for the exchange, causing a temporary jump into task space, before returning to discourse space (e.g., schematically illustrated as transition X in Figure 1).

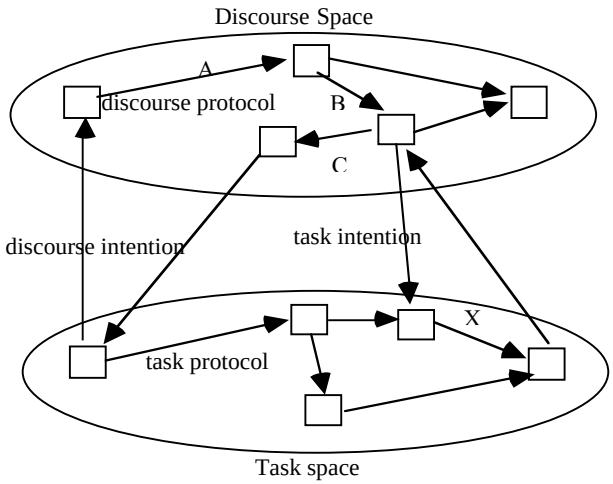

Figure 1: Schematic Relation between Task and Discourse State Spaces

Protocols are the realizations of operators in each of these two spaces. A protocol defines a structured exchange of information between two agents. Task protocols correspond to task operator for computing a successor state. (In our application they also represent communication between an agent that collects information from the user (agent) and an agent that performs computations). A task protocol is defined as a set of input parameters, that include a given state, and a set of output parameters, which fully define the successor state. We hasten to add that in most cases, this can be modeled as a simple function call, but we adopt the protocol terminology for consistency in our view of information exchange to promote state transitions.

Discourse protocols advance discourse intentions and move the agents through discourse space, which includes shared knowledge. They specify temporal and contextual conditions on

what message performatives, defined in some ACL, may follow each other. The semantics underlying the message primitives used in our framework, like others, borrow heavily from speech act theory [23], and the semantics are based on a number of pragmatic principles discussed in Haddadi [12,13]. The syntax of our messages is not our focus here; examples can be found in [6]. Briefly, a message consists of a specific message or performative type, a specific object of discourse, and a partially specified content. It has the following general format, which has much in common with most ACL message primitives: *(performative $agent-name1 $agent-name2 $object-of-discourse $content).* The *$agent-name1* parameter refers to the speaker, sender or generally the actor of the performative, while *$agent-name2* refers to the hearer, receiver or generally the agent that would be effected by the performative. The object of discourse and content flow to the individual performative via the discourse intention from the task intention that caused this protocol to execute.

In the DB agent, we make use of three classes of performatives request, query, and inform which we further specialize by specifying an inner performative that supplies information related to the result of the task action that must be performed, the task itself, or the action that the speaker intends/expects the hearer to perform. The third column in Table 1 designates the performative that would "complete" the protocol initiated by the performative in the first column. Simply put, Table 1 defines basic state-transition definitions for sub-dialogues of very short length.

Table 1

Class	**Specialization**	**Expected reply**
Request	provide	inform - provide
Request	suggest	inform-accept inform reject
Query	provide	inform-provide
Query	confirm	inform-confirm inform-deny
Inform	provide	acknowledge

Following Haddadi [13], we view request has having an associated level of commitment. Briefly, a request performative is tightly coupled with advancing the task via specification of information that will enable a task action to occur and hence the objects of discourse for this performative are those defined by task operator preconditions. The expression of commitment via a particular request protocol flows directly from roles that agents can have in terms of functionality as defined in the abstract task specification. Under the abstract task model for our simple search task, the DB agent expresses a precommitment to advancing that task in a particular manner and communicates with the user the necessary information it needs from the user to do so. By supplying the expected inform, the user is committed to the computation. The asymmetry in the functional roles assumed by the agents in the task model dictates which discourse protocols they follow in expressing these intentions. The DB agent's precommitment to advancing the task in a particular way (e.g., a particular constraint to pursue) comes in the form of a request-suggest, which is accepted or rejected by the user agent. When the user supplies information that would enable a task action to be taken, the user is simultaneously committing to a particular computation (a task intention realized in a particular manner) and delivering the necessary information to do it. When coming from the user agent, such a request is associated with a request-provide protocol. Crudely put, this is a command. Our aim is to allow the nuances of command vs. suggest, despite having been expressed by the same performative, to flow from the explicit representation of agent roles at the abstract task specification level.

Queries and informs do not advance the task directly but exchange information about the dynamic and static aspects of the task we outlined below. Hence, the objects of discourse for inform and query are dynamic task features, as delimited by the task operator preconditions and post condition. They are further constrained according to the "must know" or "may not know" aspects of the task we discussed earlier.

We have omitted much of the abstract task model's specification and its ontology, choosing instead to motivate its need and role in designing agents. For more details on this model, see [6].

5. IMPLEMENTATION

We have built the agent-assistant for successive refinement search using this abstract task model using the PRS-CL [18] architecture. The abstract task model itself as part of a conversation policy would say nothing about implementation, but rather would define the set of possible intentions and the conditions under which they would arise. That said, there are few things to be learned from understanding how the theoretical model as described above was realized in a working system. Here, we emphasize here how the abstract task specification supports a pragmatic and flexible handling of "unexpected" messages those not sanctioned as immediately expected responses for a running discourse protocol. The abstract task model also specifies the content of the performatives that are exchanged within discourse protocols. We also comment on the matter of managing the task vs. discourse space movement within this particular BDI framework.

Generally put, a BDI architecture consists of a dynamic (working) memory that holds current goals and beliefs (symbolic patterns), a plan or act (we use the term plan and act interchangeably) library for holding representations of methods and actions to achieve goals, and intention structures that correspond to plans that are in some state of execution or activity. On any given cycle, the patterns in dynamic memory trigger invocation conditions of the plans. An instance of any such activated plan corresponds to an intention and each such activated intention serves as the "root" of a new intention structure. There may be many such intention roots corresponding to different instances of the same or different invoked plans. By some resolution scheme (provided by the architecture or imposed by the designer), one act instance is selected and its associated plan begins to execute. It is possible that the execution of one plan invokes instances of other plans in the plan library, and these are managed as sub-goals or sub-plans of the calling intention. Execution of the plan or its completion typically cause changes to dynamic memory, so that other plans may be invoked, leading to new instances of acts (new intentions We note there that a plan may be suspended and never necessarily resumed. The dynamic memory also receives external events and a message from another agent would be such an event.

Within this architecture, we have functionally partitioned dynamic memory into a task-space and discourse space. The task space is modeled as a stack: the current task state is the top of this stack. The discourse space is updated with new information that corresponds to the agent's shared task with the other agent, as well as information about currently-active discourse protocols (discussed shortly).

Both task intentions and discourse intentions are represented as plans in the plan library. We distinguish between DB-intentions and user (or other agent) intentions. DB-intentions are the intentions the DB agent sets, and user-intentions are the intentions the DB agent recognizes via the arrival of a message with a particular content. We will first step through the setting of a DB intention, and then speak to the matter of recognizing user intentions.

Conceptually, a plan has an invocation condition, resumption conditions, and satisfaction conditions. Invocation conditions are those conditions that trigger the plan and form an instance of an intention, i.e., constitute a goal that has been adopted. If an intention's satisfaction condition is not met, the plan's body specifies a method for trying to satisfy the satisfaction condition. For example, in our implementation there is a DB task intention (plan) corresponding to *contract-the-search-space*. An instance of that plan becomes created and active when the current task-space state has certain properties. The satisfaction conditions correspond to the preconditions for actually contracting the search-set (e.g., having a new constraint completely specified). The satisfaction conditions for a task intention correspond to what we have earlier called task-operator preconditions. The body of this task intention the plan invokes discourse intentions as the means by which these satisfaction conditions are satisfied. There may be several possible discourse intentions associated with the achievement of any particular satisfaction condition of a task intention. For example, the DB agent may obtain information from the user agent by asking for it or by suggesting a possibility. The selection is driven by task and discourse context patterns.

The invoked discourse intention's own method is a discourse protocol. Discourse protocols are also modeled as plans, but these are plans for conversation sub-units. Through its input parameters (its activation conditions), a discourse protocol receives all the specific variable bindings it need to formulate the content of the performative it will use in its starting message; this information has migrated down to it from the originating task intention through the discourse intention. The plan bodies for all discourse protocol are the same: the protocol formulates the message, sends the message out, posts its expected reply to dynamic memory with an appropriate protocol ID tag, and then suspends.

When a message arrives, a message handler also modeled as a plan looks to see which protocol is waiting for it (there might be many suspended threads of conversation). The most recently suspended protocol expecting that message type receives it. This resumes execution of the discourse protocol, whose successful completion causes updates to discourse space. Resumption of the discourse intention ensues, and its completion causes transfer of information from the new discourse space-state into the current task-space state. The task intention then resumes to see if all its satisfaction conditions are satisfied. If not, its plan body may set another discourse intention.

What happens if the incoming message is not what any currently-suspended discourse protocol expects, i.e., it is an undefined transition for message exchange? The message handler, having established it goes to no waiting protocol, releases it into dynamic memory. There, it triggers the *recognition* of a user intention from within the plan library. In our framework, to recognize an intention from the other agent is to recognize that a new plan, valid within the confines of the task specification, is being followed. For example, all queries from the user are dealt with in this manner and recognized as user discourse intentions. This is in contrast to another common approach, namely to specify for each and every performative whether or not a query message is valid next-transition. This makes the local and extended effects of a protocol on both discourse space and task space much clearer. That is, the arrival of a message that is unexpected in the local context of an executing discourse protocol is resolved at the global level of possible discourse intentions or task intentions.

Dialogue systems require some dialogue management algorithm. In our implementation, the functionality of this realized via an activation priority scheme that applies to invoked plan instances. First, the message handling plan is given highest priority: incoming messages are resolved against a waiting discourse or released to trigger the recognition of a user task intention or user discourse intention. Second, user-intentions (again, realized as instances of plans from the plan library) take precedence over any intention set by the DB agent.. And activated discourse intentions are given priority over activated task intentions. We hasten to note that this is sort of strategy constitutes a crucial aspect of the shared conversation policy that the two agents follow and can be make explicit in the task model specification, possibly as a function of their respective roles.

Our implementation handles the conversation flow illustrated by our earlier dialogue, as well as others in which there are several changes of direction, multiple sub dialogues embedded within sub-dialogues that in turn lead to direction change, and so forth. This application domain is admittedly simple: there are 24 intentions in the plan library corresponding to task intention, discourse intentions, and discourse protocols; task protocols are modeled as function calls taken as the last step of a satisfied task intention. It would be more consistent, albeit less convenient, to move them to the plan library.

6. RELATED WORK

Several researchers have specified semantics at protocol-level, realized pre-, post-, and completion conditions for the primitive performatives [4, 14, 26] There is a correspondence to these ideas to the semantics we have for our abstract task intentions and protocols, except that our task intentions concern how to advance the task, given the agent s beliefs about the current task state, and discourse protocols are then called in service of the intention. Thus, we separate our communication semantics from our task semantics. It is only the abstract task intentions that provide any coherence across the 2- or 3-message pairings that are defined by our protocols. Furthermore, our discourse protocols are ultimately executing in service of advancing a task intention, and control is returned to the level of task intention processing, regardless of whether the protocol completes in a pre-defined manner or is suspended (along with its associated intention) due to a locally-unexpected message. Most proposals for semantics at the protocol level are defining what we would call updates to discourse space variables in our framework. The extended perlocutionary effects

of a successfully executed discourse protocol, in our framework, are specified as changes first to the discourse space and then, through the discourse intention, as changes to the task space.

Another difference between the conversation policies as specified in [2] and ours stems from our handling of locally unexpected messages. Note again that our protocols are essentially message-adjacency pairs and contain very few conditional transitions. Suppose our DB agent's message is (functionally) "Do you want expensive or inexpensive French food?" and the user responds "Are there Thai restaurants near-by?" That response might be designated with a "counter-request" message primitive. But there is no getting around the fact that some inferencing has to be done, by the sending agent or the receiving agent. Either sending agent must somehow determine that it wishes to send a message that ought to be marked as counter-proposal (e.g., using a counter-proposal message primitive), or the receiving agent must correctly interpret a vanilla "inform" message as designating a counter-proposal. Our treatment of such a message unpacks into two speech acts: inform-reject and request-provide. We are suggesting that the abstract task model approach, by defining objects of discourse, task and discourse intentions, provides the specification needed to support such inferences about what a particular message means in relation to the current state of the task.

Pitt and Mamdani [20] present protocol-level semantics that includes what they call an "add function" an agent's procedure for computing the change in its information state from the content of an incoming message using a particular performative uttered in the context of a particular protocol. We would include the extended effects of such an update to include changes into task space. They also envision a realization of their framework in a BDI architecture, much like we have implemented for the agent described here. Specifically, they call for the representation of conversations (what we call protocols) as plans, and the notion that beliefs and desires (goals) would initiate a conversation by starting an instance of the appropriate protocol. We have pragmatically realized many aspects of their protocol-based semantics via our abstract task specification, which designates the relationship between task intentions, discourse intentions, and ultimately, well-structured communication acts.

Implicitly or explicitly, performatives or message types are regarded as transition arcs or operators in some kind of discourse space. For example, many proposals for ACL semantics based on joint intention or mutual belief theory specify axioms for what intentions or beliefs are jointly held, given a particular message exchange [5, 24, 26]. That message sequence or protocol is then serving as a transition arc between two distinct states of shared knowledge. Arcs also represent the receipt and sending of messages in [1]. Here, the grain size of our transition arcs in discourse space is set at the level of the protocol, not the performative. It is only the successful completion of a protocol a structured message exchange that can have extended perlocutionary effects into task space.

Our abstract task model as part of a conversation policy shares much in common, at least in spirit, with the shared-plan notion that characterizes certain approaches to discourse processing [e.g., 11]. Here, we are proposing that an abstract task specification, explicitly defined and provided as part of a publicly posted conversation policy, serves the same purpose in delimiting the set of possible intentions. It is unclear to us how many actual agent applications will require complex semantics about joint intentions, and mutual beliefs. But it seems that they will minimally require a specification of what the joint task is, what is done to accomplish it, who knows what, who needs to know what, relative functionalities, and so forth. We have tried here to indicate how these can be formulated in a principled and pragmatic way via an abstract task specification.

7. CONCLUSIONS AND DIRECTIONS

This paper presents the notion of an abstract task specification as one aspect of a publicly posted conversation policy. We start top-down with the task specification in guiding us to the pragmatic meaning of message protocols. We define an abstract task as having a distinct characterization of knowledge types and operations describes the problem solving behavior of an agent in a variety of domains. More simply, a task is a problem type, such as diagnosis, design, or successive refinement search.. The ontology of knowledge types and operators for the abstract task is used to formulate a state-space representation of the problem. That representation defines the set of task intentions, which are satisfied via the application of task operators. Preconditions and post-conditions of task operators circumscribe the set of possible discourse intentions. Message exchange according to discourse protocols ensues to satisfy discourse intentions, which in turn enable task intentions to be advanced. Exactly which preconditions and post-conditions of task operators must or must not be objects of discourse between two agents depends on the functional roles the agents share in achieving the task. We have argued for a clean separation between discourse and task spaces, believing that it clarifies, using Moore's [17] distinction, what the "local" and "extended" effects of a message exchange are. For us, local effects of sending a message or completing a message exchange via a successful protocol are effects on the discourse space. The extended effects are those changes in shared knowledge in discourse space that impact the general task the agents are trying to accomplish in the task space.

We are presently considering a suitable representation language for presenting an abstract task specification as a downloadable conversation policy. But the crucial issue is to evaluate just how far an abstract task specification goes in allowing agents, implemented in different ways within different frameworks, to nonetheless communicate effectively on a joint task. Simply put, could two agent designers, given just an abstract task specification, create agents that can communicate effectively and achieve a task? We particularly need to examine the matter of agent roles and functionality within the abstract task specification. Our initial test domain has made several elements easy for us and further work needs to be done to generalize our approach to abstract tasks specifications in which agents roles are more balanced in terms of leadership. In particular, the asymmetry of the agent roles here eliminates the need for agreement on what to do next, what some researchers have investigated as negotiation [24]. In our view, we would like to consider negotiation as a task with its own operator set and state space, again subservient to the main task (diagnosis, design, search) that the agents are cooperating to accomplish. If one is committed to this view, it is unclear that a BDI architecture is best suited for managing problem-solving in distinct spaces, with distinct plan libraries. Instead, we are considering architectures like SOAR [16], in which an "impasse" in one problem space (the task space) spawns a new problem space (the discourse space).

In sum, we advocate a pragmatic approach to determining intentions based on task specifications that agents may jointly share. In doing so, this approach serves as a way of constraining what messages agents send to each other at the task level and reduces the burden of imposing all such constraints at just the individual performative or at just the protocol level. This is, of course, just one step towards coherent message exchange, based on "intentional states" that can be directly and easily traced to the global task.

8. ACKNOWLEDGEMENTS

This work was supported in part by NSERC Grant A0089 to R. Elio and continuing support by DaimlerChrysler to A. Haddadi.

9. REFERENCES

[1] Barbuceanu, M. and Fox, M. COOL: A language for describing coordination in multi-agent systems. In Proceedings of First International Conference on Multi-Agent Systems, 1995.

[2] Bradshaw, J. M., Dutfield, S., Benoit, P., and Woolley, J. D. (1997). KAoS: Toward an industrial-strength open agent architecture. In J. M. Bradshaw (ed.), Software Agents, AAAI Press, Menlo Park CA, 375-418.

[3] Bylander, T. and Chandrasekaren, B. Generic tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge engineering. International Journal of Man-Machine Studies, 26, (1987) 231-243.

[4] Cohen, P. R. and Levesque, H. J. Communicative actions for artificial agents. In J. M. Bradshaw (ed.), Software Agents, AAAI Press, Menlo Park CA, 419-436, 1997.

[5] Cohen, P.R. and Levesque, H. J. Intention is choice with commitment. Artificial Intelligence, 42 (1990) 213-262.

[6] Elio, R. and Haddadi, A. On abstract tasks and conversation policies. In M. Greaves & J. Bradshaw (eds). Specifying and implementing conversation policies. Autonomous Agents '99 Workshop (Seattle WA, May 1999), 89-98.

[7] FIPA-99 specifications, see www.fipa.org/spec

[8] Greaves, M. and Bradshaw, J. (eds.). Specifying and Implementing Conversation Policies, Autonomous Agents '99 Workshop, (Seattle, WA, May 1999).

[9] Greaves, M., Holmback, H., and Bradshaw, J. What is a conversation policy. In M. Greaves & J. Bradshaw (eds). Specifying and implementing conversation policies. Autonomous Agents '99 Workshop (Seattle WA, May 1999), 1-10.

[10] Grice, H.P. Logic and conversation. In P. Cole & J.L. Morgan (eds.) Syntax and Semantics, Vol 3: Speech Acts. Academic Press, New York, 225-242, 1975.

[11] Grosz, B.J. and Sidner, C.L. Plans for discourse. In P. Cohen, J.L. Morgan, and M.E. Pollack (eds.) Intentions and Communication. MIT Press, Cambridge, 417-444, 1990.

[12] Haddadi, A. Towards a pragmatic theory of interactions. In M. N. Huhns and M. P. Singh (eds.) *Readings in Agents*, Morgan Kaufmann, San Francisco, 443-449, 1998.

[13] Haddadi, A. Communication and cooperation in agent systems: A pragmatic theory. Lecture Notes in Computer Science, #1056, Springer Verlag, Berlin, 1996.

[14] Labrou, Y. and Finin, T. A semantics approach for KQML A general purpose communication language for software agents. In Third International Conference on Information and Knowledge Management, ACM Press, New York, 447-455, 1994.

[15] Labrou, Y. & Finin, T. Semantics and conversations for an agent communication language. In M. N. Huhnhs and M. P. Singh (Eds.) Readings in Agents, 235-242. Morgan Kaufmann, San Francisco, 235-242, 1998.

[16] Laird, J.E., Newell, A., and Rosenbloom, P.S. SOAR: An architecture for general intelligence. Artificial Intelligence, 33 (1987) 1-64.

[17] Moore, S. On conversation policies and the need for exceptions. In M. Greaves and J. Bradshaw (eds). Specifying and implementing conversation policies. Autonomous Agents '99 Workshop (Seattle WA, May 1999), 19-29.

[18] Myers, K. A procedural approach to task-level control knowledge, Proceedings of the Third International Conference on AI Planning Systems, 1996.

[19] Phillips, L and Link, H. The role of conversation policy in carrying out agent conversations. In M. Greaves and J. Bradshaw (eds). Specifying and implementing conversation policies. Autonomous Agents '99 Workshop (Seattle WA, May 1999), 11-18.

[20] Pitt, J. & Mamdani, A. Communication protocols in multi-agent systems. In M. Greaves and J. Bradshaw (eds). Specifying and implementing conversation policies. Autonomous Agents '99 Workshop (Seattle WA, May 1999), 39-48.

[21] Reinhart, T. Pragmatics and linguistics: An analysis of sentence topics. Philosophica 27, (1981)53-94.

[22] Rich, C. & Sidner, C. L. COLLAGEN: When agents collaborate with people, In M. N. Huhns and M. P. Singh (Eds.) Readings in Agents. Morgan Kaufmann, San Francisco, 1998, 117-124.

[23] Searle, J.R. Speech Acts. Cambridge University Press, New York, 1969.

[24] Sidner, C. L. An artificial discourse language for collaborative negotiation, *Proceedings of AAAI* (1994) 814-819.

[25] Singh, M.P. Agent communication languages: Rethinking the Principles. IEEE Computer 31 (1998), 40-49.

[26] Smith, I. A., Cohen, P.R., Bradshaw, J. M., Greaves, M., and Holmback, H. Designing conversation policies using joint intention theory. In Proceedings of the Third International Conference on Multi-agent Systems, 269- (Paris, France 1998) IEEE Press, 269-276.

[27] Winograd, T. and Flores, F. Understanding computers and cognition. Ablex Press, New Jersey, 1987.