Elio, R. & Haddadi, A (1999). Abstract tasks and conversation policies. In F. Dignum & M. Greaves (Eds.) *Issues in Agent Communication*, 301-314. Springer Verlag, Berlin

On Abstract Models and Conversation Policies

Renée Elio[1] and Afsaneh Haddadi[2]

[1]Department of Computing Science, University of Alberta,
Edmonton, Alberta Canada, T6G 2H1
ree@cs.ualberta.ca
[2] DaimlerChrysler, AG Alt-Moabit 96A,
10559 Berlin, Germany
afsaneh.haddadi@daimlerchrysler.com

Abstract. It is possible to define conversation policies, such as communication or dialogue protocols, that are based strictly on what messages and, respectively, what performatives may follow each other. While such an approach has many practical applications, such protocols support only "local coherence" in a conversation. In a mixed-initiative dialogue between two agents cooperating on some joint task, there must be a "global coherence" in both the conversation and in the task they are trying to accomplish. Recognition of agent intentions about the joint task is essential for this global coherence, but there are further mechanisms needed to ensure that both local and global coherence are jointly maintained. This paper presents a general yet practical approach to designing, managing, and engineering agents that can engage in mixed-initiative dialogues. In this approach, we promote developing abstract task models and designing conversation policies in terms of such models.

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, 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 [9]. Our concern here is what some

researchers [3] 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. 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 communication language, such as KQML or FIPA's ACL [2,7]. 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 [2, 6, 10,15]. 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 [16].

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 Greaves, Holmback & Bradshaw [3] note, 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. 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 third matter. Focusing on protocols alone will not fully address the primary function of a conversation policy as motivated in [3]: 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 message exchange?

The appeal to global coherence as a feature of a conversation is implied by Grice's [4] 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 [12]. 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 [1]. Further, the notion of a “shared plan” has a prominent role in models of human discourse processing [e.g., 5]. 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 performatives 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, its associated intention cannot be undefined in the context of the abstract task specification that is jointly held by the two agents.

As one component of a publicly posted conversation policy, an abstract task model addresses two elements of a broadly-defined conversation policy [3]: specific goal achievement policies and some aspects of conversation management policies. We illustrate the features of this approach by showing its application to a simple and well-understood agent-assistant task.

2 Successive Refinement Search as an Abstract Task

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, which we here call the database agent (DBA), 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 DBA 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 assume that the database is so that the user cannot specify a single, simple database query (e.g., "Where is the closest gas station?") 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 DBA and the user as two agents 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 illustrates the nature of the task, where U designates the user.

DB-1: What type of location do you want to locate?
Lumber stores? Pharmacies?
U-2: I want inexpensive food in San Francisco.
DB-3: O.K.. There are several hundred inexpensive restaurants in San Francisco.
DB-5: What type of food do you want?
U-6: What do you mean?
DB-7: I mean ethnic type such as Hungarian or French.
U-8: Italian.
DB-9: O.K. Do you want a take out or sit-down place?
U-11: What about on the Peninsula?
DB-12: O.K. We'll leave San Francisco for a moment.
DB-13: There are 35 Italian restaurants on the Peninsula.
DB-14: A particular place or city will rapidly narrow
down this set most rapidly.
U-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 [13]).

First, in viewing the above dialogue to be about successive refinement search (and not about restaurant advising or locating), it appears there are a few generic objects of discourse that comprise the message content:(i) a *domain*: a particular domain database in which entities are defined by features and values; (ii) a *constraint*: any feature that has a particular value assigned to it; (iii) a *search space*: a set of database entities that satisfy a set of constraints; (iv) *search-space members*: particular entities within a particular search space.

From our perspective, the semantics underlying the messages being exchanged in the example dialogue are defined by the ontology of successive refinement search as an abstract task. What these objects of discourse are and what can be said about them are intimately defined with what actions can be taken to advance the task. In our analysis these actions are limited to: (i) *loading* a database to be searched, which defines the initial search space, (ii) *contracting*, or reducing the search space by specifying additional constraints that members must satisfy, and (iii) *expanding* that search space by relaxing one or more constraints. These are traditional database operations. There may be other capabilities that are unique to each agent (e.g., an agent might also compute the most-discriminating feature for a given search space.)

Another key aspect about the sample dialogue above is that either agent can take the initiative in advancing the task in a new direction. Therefore, the DBA must respond effectively to these "unexpected“ messages. For example, in utterance U-11, the user does not provide an answer to the question posed in utterance DB-10, and instead shifts the direction of the search task. Intention recognition serves to support this mixed-initiative aspect of cooperation. Intentions that an agent can have about the task (and presumably express during the conversation) are limited by the objects of discourse, what can be done with them, and therefore what can be said about them. This is crucial to having a pragmatic but somewhat flexible approach to posting and recognizing intentions, for these objects of discourse serve to circumscribe the intention set.

3 The Abstract Task Defines Objects of Discourse

The semantics underlying the language primitives used in our framework borrow from general speech-act theory and the semantics are based on a number of pragmatic principles discussed in Haddadi [6]. A message consists of a specific message type (which we will call a performative type), specific object of discourse, and partially specific content. It has the format *(performative $agent-name1 $agent-name2 $object-of-discourse $content).* For brevity's sake, we have omitted many message parameters such as feature keywords and others for protocol administration.

Table 1 presents the set of performatives defined for the DBA. The outermost performatives represent the general class of an utterance. In the DBA, we make use of the classes request, query, and inform. The inner performatives given in Table 1 further specialize the class, by supplying information related to the result of the task action that has been performed, the task itself or the action the speaker intends/expects the hearer to perform. 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.

Table 1. Performatives and their Combination

Outer Performative	Inner Performative	Immediately Expected Reply
Request	Provide	Inform
	Suggest	Inform + Accept/ Reject
Query	Provide	Inform
	Confirm	Inform + Confirm/Deny
	Suggest	Inform + Accept/Reject
Inform	Provide	
	Confirm	
	Deny	
	Accept	
	Reject	

The third column of Table 1—the immediately expected reply—designates the performative that would "complete" the dialogue initiated by the performative in column 1. Put another way, the information in Table 1 defines a basic state-transition definition for sub-dialogues.

Inform. Inform messages take on the outer and inner objects-of-discourse, and a content specification, that occur in the request or query dialogue that they complete. *Request*. A request performative is tightly coupled with advancing the search task. The objects of discourse associated with request are (i) a system *action* that enables a search task to begin or terminate, such as loading a particular database for searching and (ii) a *constraint*, which specifies a feature-value vector according to which database entities can be identified. Most request performatives concern constraints.

Following [6], we view a request as having an associated level of commitment. When a request is made by the DBA, the system is making a pre-commitment to how the progress on the search task might be accomplished and it prompts the user for information in order to do this. System requests thus take *suggest* as an inner

performative. A suggestion refers to a possible task strategy and it must be accepted or rejected by the other agent, in this case, the user. By supplying the information asked for, the user is committing to this computation. When a request is made by the user, the user is simultaneously committing to a computation on the search space and delivering the information necessary to execute it. User requests thus take *provide* as an inner performative. We note that the abstract task model is the place to specify agent roles with respect to functionality and this in turn can be used to derive the appropriate nuances of message types such as "command" versus "suggest." Some examples of request messages, with their English gloss, include the following (where U means user-agent and S means system-agent)

(i) ReqU: Lets look for a restaurant in Mid-Peninsula.
(request U S :constraint (provide U S :value (fv-pairs :feature location:value Mid-peninsula)))

(ii) ReqS: How about Chinese?
(request S U :constraint (suggest S U :value (fv-pairs :feature rest-type:value Chinese)))

(iii) ReqS: What kind of price range?
(request S U :constraint (provide U S :value (fv-pairs :feature price :value ?)))

Query. A query is not about advancing a task but about exchanging information indirectly related to advancing the task. An abstract task model defines an ontology of what each agent knows about as well what each agent may be *allowed* to know about. This circumscribes to some extent the message content of queries; for the successive refinement task, the objects of discourse are: (i) system *functionality*, that is in what domains it can assist the user with the search task (ii) the *domain knowledge base*, which includes domain-specific information, such as the range of values on a particular feature, (iii) the *database*, which includes queries about the availability of information about particular entities in the database, and (iv) *task-information*, information relevant to the current state of the task. Queries may take either provide, suggest, or *confirm* as an inner performative. A confirm expresses the truth or falseness with respect to a property of some object-of-discourse and it must be confirmed or denied. When a query is sent by the user agent, the system must respond with an *inform* followed by an appropriate inner performative. System queries often take the form of suggestions, in which case the user response must be interpreted as providing (at least) an acceptance or rejection of the suggestion. The objects-of-discourse that may accompany the inner performatives associated with queries are restricted to (i) the *domain* (e.g., restaurants, hospitals); (ii) the current *search space* (i.e., the set of members defined by the current set of constraints); and (iii) a

particular *member* in the current search space or database. Examples of query messages with English correspondences include the following:

(i) QueU: Do you have menus for restaurants?
(query U S :knowledge-base (confirm S U :domain
(describe-domain :domain $domain (has-feature :feature menu)))

(ii) QueU: Do you have the menu for Maxine's?
(query U S :database (confirm S U :member (has-attribute :member
$member :feature menu)))

(iii) QueU: What do you mean by (the descriptor) restaurant type?
(query U S :know-base (provide S U :domain (describe-feature
:domain $domain-id feature-list :feature rest-type :attribute range))

We now have a representation and semantics for the abstract task model of successive refinement search and have covered all possible task intentions (i.e., intentions in the sense of illocutions associated with speech acts) that agents can express when communicating about the search task.

We cannot overemphasize the role of the abstract task model in specifying the ontology for these performatives. The successive-refinement task model defined a semantics for the general performatives in terms of what could be talked about. A more general theory of semantics for performatives in terms of what messages must or may follow each other is found in [6]. However, the content of the performatives, i.e., the objects of discourse and what can be said about them, can only follow from the joint commitment to a shared abstract task model. What we have developed here is an abstract task analysis for cooperative successive refinement search, and let that abstract task model define these objects, their relations, and the set of methods that allow progress to be made on the task.

4 Abstract Tasks Define Intentions and Global Coherence

At this point, we have only covered utterances and their associated intentions in communication (i.e., the illocution of an utterance). Table 1's specifications about what performatives may or must follow each other ensure some local coherence at the level of 2-3 message sequences, but they do not structure how short message sequences can be patched together in a globally coherent way. We now consider how intentions are internalized for reasoning about the next course of action (progress on the task vs. progress in the discourse) and how to maintain a coherent dialogue between agents (i.e., coherent perlocutions). We assume a turn-taking dialogue. The

abstract task model we apply here allows (i) that both the agents can take the initiative in directing the task and (ii) the user's initiative always has priority over the system's initiative. While the user may change the direction or other aspects of the search at any time, the abstract task model defines that the system would do so only if the current direction could not be pursued further. In this case, the system takes initiative in suggesting a new course of action. The abstract task model, by specifying strategies for task progress (such as identifying the most discriminating feature to reduce a search set), also defines a larger set of intentions that the either agent can have,

There must be (i) a means for deciding or recognizing that a particular message is relevant to advancing either the task, the exchange of information about the task, or switching to a completely new task, and (ii) a means for representing how the semantics of the performative pairs in Table 1 actually advance these goals. The first is accomplished by relating agent intentions about the task or the discourse to specific performatives. The second is accomplished by protocols, which explicitly map sequences of messages to the fulfillment of a particular intention, and indicate what must happen when an unexpected message is received. We now turn to a more detailed discussion of intentions and protocols, and how they are represented.

4.1 The Interplay of Task Space and Discourse Space

In previous sections, we have underscored that the DBA must reason about both the task of successive refinement search and about the task of structuring the discourse with the user in an intelligent way. These two tasks are not only related, but at different times, one supports the other. Here, we introduce the conceptual distinction between "task space" and "discourse space", and indicate how intentions and protocols define elements of these two spaces.

Let a state in the discourse space correspond to shared information held by both the DBA and the user. A transition between one state into another is made by making an utterance (e.g., "Where do you want to eat?"). When an utterance is correctly interpreted, its intention is understood and that intention becomes part of the shared information between the two agents. For the abstract task of successive-refinement search, the problem at the task level is to identify a set of entities in a database that meet certain constraints. A state in this task space is a set of database entities and the constraints that define them. There are two legal transitions or operators at this level: the contract operator, which means adding a new constraint, or the expand operator, which means relaxing an existing constraint. When either of these operators are applied within a particular state, a new state results, defined by a new set of database entities and the feature-value specifications that uniquely define them. The goal state

in the task space is the set of database entities that the user deems to be sufficient for his or her purposes.

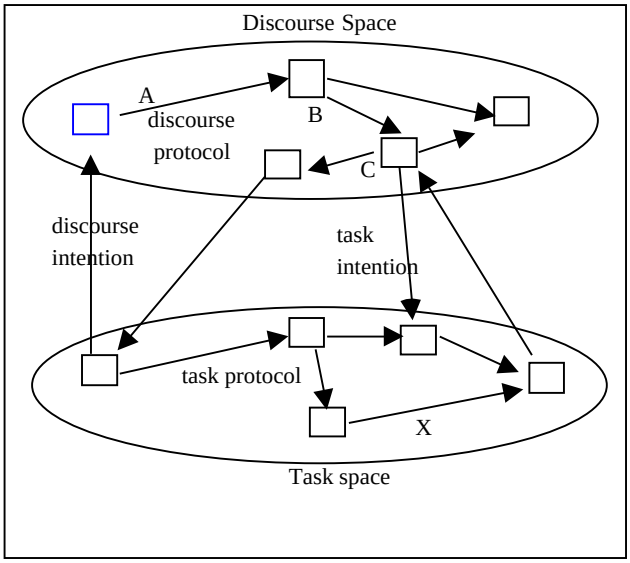

Fig. 1. Schematic Relation between Task and Discourse State Spaces

Under this perspective, an intention is commitment to act to achieve a particular state in one of these spaces. A protocol is the realization of operators in each of these two state spaces. The interplay of task space and discourse space is summarized in Figure 1. The crucial feature is this: whenever an agent cannot proceed in one space, it must form an intention to make a transition in the other space. Thus, the failure to move ahead in the task space (e.g., contract the search space of entities) requires an intention to make a transition in the discourse space (e.g., a request for or suggestion about possible constraints, schematically illustrated as transitions ABC in Figure 1). Conversely, a transition in the discourse space may require movement in the task space (e.g., some new computation on domain objects to identify the answer to a query, schematically illustrated as transition X in Figure 1). We argue for a clean separation between discourse and task spaces, believing that it clarifies, using Moore's [8] 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

4.2 Intentions and Protocols as Plans

An intention to achieve a goal is represented as a general plan to advance either the task (in which case it is a task intention) or the exchange of information (in which case it is a discourse intention. We have adopted a generic belief-intention-desire framework, in which an intention is represented via (i) an invocation condition, which returns true if the intention is applicable to the current task context; (ii) a completion condition, (iii) a method-to-advance plan; and (iv) a resumption condition. The completion condition determines whether there is enough information in the current (task or discourse) state to satisfy the intention. If there is, a transition is made in the current space by executing one of the protocols specified in the completion routine which is appropriate in the current task context. Otherwise, the method-to-advance—the plan body— is invoked. This leads to the adoption of an intention which is often an intention to move in the other, The resumption condition determines whether the intention, if it had been suspended, can be resumed. We will return to this later.

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. In our case, discourse protocols implement the semantics for adjacency-pairs of performatives, as given in Table 1.

Protocols are also represented as plans within the belief-desires-intention framework we have adopted, whose representation has the form of (i) an invocation condition, which is the name of its invoking intention, and (ii) a plan body which we conceptually divide into invocation portion and an assimilation portion. The invocation portion of the plan body creates a message of the appropriate type and sends it. The assimilation portion of the protocol's defines the expected message and the associated state updates that would occur, if the protocol were successfully completed. After the protocol's plan sends its message and identifies its expected response, it is suspended.

It is in the realization of this assimilation routine that we allow for *locally* unexpected messages. In the case that the next arriving message is not the one

expected by the most recently suspended protocol, that protocol remains suspended and the message is mapped to the intention plan library, where it triggers the recognition of a user intention. This is the manner in which the DBA can switch to a state in the task space that is not a predefined transition from its current task state. the DBA's modest amount of "reasoning" allows unexpected performatives (i) to be recognized as unexpected, (ii) to trigger the recognition of user intention and (iii) to pass control to responding to that intention. Exactly what new task intention is being signaled by the unexpected message requires a scheme for intention-matching on the part of the DBA. But the crucial role of the abstract task model, in conjunction with the current task context, is to define a small set of possibilities.

A suspended intention (which is functionally suspended if its associated protocol is suspended) may never be resumed, if a new intention causes a state change in task or discourse space, such that the suspended intention has become irrelevant. Here is where the distinction we make between request and query performatives play an important role. Requests are directly associated with transitions in the task space and a commitment to making those transitions. Hence a user request causes adoption of a new task intention. If the completion condition of this task intention is true, the course of the current task context will change and this will be reflected in the task context variables. The resumption condition of a suspended intention checks these variables to determine if the intention may be resumed or abandoned. We require that a new direction in the search task results in abandoning a suspended task intention and its associated protocol. If the user performative, however, was a query (causing a transition in discourse space), the suspended intention (in task space) may be resumed.

4.3 Conversation Management

We have implemented the above design in PRS [see 11], a belief-desires-intention architecture, in which both intentions and protocols are implemented as plans. Some aspects of conversation and task management are implicitly handled by the architecture itself, such as the passing of control from one plan to another. Other aspects of task and conversation management are a realized as plans as well. On the abstract level, given the turn-taking assumption in dialogue, the general task and discourse management algorithm follows the following priority scheme: (i) recognize the need to set or respond to discourse intentions, (ii) react to any user intentions, and (iii) set and pursue the DBA task intentions. DBA task intentions are set and pursued through the combination of an intention priority scheme (defined by the abstract task model) that indicates how features of the current task context trigger certain types on intentions. We note that this sort of dialogue (or conversation) management algorithm

characterizes much of the work in the discourse processing area. What is new here is our reliance on formalizing an abstract task model to implement the crucial aspects of run-time intention recognition in a practical and domain independent way.

6 Discussion

The framework we have described could be viewed as a two-level state-transition model: the abstract task model defines transitions at the level of task intentions and the discourse protocols define transitions at the level of shorter sub-conversation messages. Task intentions essentially structure the problem solving process and in turn, the dialogue that can ensue about the problem solving process. However, we allow that certain task intentions may be forever abandoned, as the initiative between agents shifts during the problem solving process. Hence, we allow for transitions as well as "jumps" between intentional states. It is through this sort of approach that we aim to realize a pragmatic implementation of notions such as beliefs and intentions. While mixed initiative dialogues are currently mostly studied in relation to interacting with users, we envision mixed initiative dialogues between software agents to be an important requirement for many applications systems of the future. In particular, in complex settings such as agent to agent negotiation, where protocols as they are typically employed simply cannot provide sufficient flexibility needed by agents for exploring the negotiations space.

Pitt and Mamdani [10] 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 DB 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.

The matter of adopting the idea of abstract tasks to sub-tasks within a larger task needs further consideration. The first point is whether our notion of a protocol "violation" is too restrictive a mechanism for signaling an intention change and thus moving to some different subtask within the problem. When a subtask is completed, the agents will have to naturally move onto some new intention to make further

progress. We do not intend that protocol violations are the only means of shifting attention. A violation merely means that the current intention may need to be dropped. In our brief discussion of conversation management, we noted that the algorithm follows a three-step priority scheme—the last step is "set and pursue intentions." Those intentions in turn are specified by a strategy that might be viewed as a meta-plan. This intention-ordering scheme is somewhat like a plan of attack that sets intentions as a function of they dynamic task context. A hierarchical task network planning approach is likely to provide further structure for very complex tasks. We use protocol violations merely as a way to signal something about the current intention: a message that cannot be resolved in the context of the currently active protocol must be signaling a different intention.

A second issue concerns the dominance of the user in the approach we have advocated here. On the one hand, we are calling this a 'mixed initiative approach', which makes it seem as if both agents have equal status. On the other hand, the DBA's intentions take back seat to any user intention. Note that this is defined strictly within the discourse management algorithm and not within the abstract task model. If both agents are software agents, then the question naturally arises about which agent steers the conversation, or whether they both would be searching inefficiently for sub-conversations. Two agents have to cooperate because they each have access to different sorts of knowledge and different capabilities. Even if the two agents shared the same abstract task model (and by definition, this would include the same strategy or plan of attack outlined in the previous paragraph), one of them must know something or be able to do something the other cannot. (In our case, the human agent has the constraint information and the goal information, and the DB agent has the database access and system functionality to compute the results). However, we believe that the abstract task model is the place for defining agent roles and functionality, thus clarifying the range of task management and discourse intentions for each agent.

The critical, more general issue concerns how to assess the pragmatic advantage that may be gained by specifying the semantics for abstract tasks as the semantics that are required to support flexible multiagent communication. Within the knowledge-based system arena, Chandrasekaren and his colleagues advanced the notion that complex tasks could be decomposed into sub-tasks that were, in some sense, generic, such as classification, data retrieval, plan selection and refinement, and so forth. If there are generic ontologies associated with generic tasks, then that ontology can be the foundation for intentions that in turn can structure the conversation and the content of the individual performatives. Thus, a crucial area for further work is assessing whether and how this abstract (or generic) task approach can be leveraged

to define semantics for a core set of objects-of-discourse to be used in the evolution of some standard task languages for agents.

Acknowledgments

Aspects of this work were developed while A. Haddadi was seconded to the Adaptive Systems Laboratory, Daimler-Benz Research and Technology Center, Palo Alto, California, where R. Elio spent part of a sabbatical during 1997-1998. The support of that group, the University of Alberta, and NSERC Grant A0089 to R. Elio is gratefully acknowledged.

References

1. Bylander, T., 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
2. FIPA-99 specifications, see www.fipa.org/spec
3. Greaves, M., Holmback, H., Bradshaw, J.: What is a conversation policy. In Dignum, F., Greaves, M. (eds.): Issues in Agent Communication. Springer-Verlag, Berlin Heidelberg New York (2000)
4. Grice, H.P.: Logic and conversation. In Cole, P. & Morgan, J.L. (eds.): Syntax and Semantics, Vol 3: Speech Acts. Academic Press, New York (1975) 225-242
5. Grosz, B.J., Sidner, C.L.: Plans for discourse. In Cohen, P., Morgan, J.L., Pollack, M.E. (eds.): Intentions and Communication. MIT Press, Cambridge (1990) 417-444
6. Haddadi, A.: Communication and cooperation in agent systems: A pragmatic theory. Lecture Notes in Computer Science, Vo1.1056 Springer-Verlag, Berlin Heidelberg New York (1996)
7. Labrou, Y., Finin, T.: Semantics and conversations for an agent communication language. In Huhn, M. N., Singh, M. P. (eds.): Readings in Agents. Morgan Kaufmann, San Francisco, (1998) 235-242
8. Moore, S.: On conversation policies and the need for exceptions. In Dignum, F., Greaves, M. (eds.): Issues in Agent Communication. Springer-Verlag, Berlin Heidelberg New York (2000)
9. Phillips, L., Link, H.: The role of conversation policy in carrying out agent conversations. In Dignum, F., Greaves, M. (eds.): Issues in Agent Communication. Springer-Verlag, Berlin Heidelberg New York (2000)

10. Pitt, J., Mamdani, A.:Communication protocols in multi-agent systems. In Dignum, F., Greaves, M. (eds.): Issues in Agent Communication. Lecture Notes in Computer Science,Springer-Verlag, Berlin Heidelberg New York, (2000)
11. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. Tech. Rep. 56, Australian Artificial Intelligence Institute, Melbourne, Australia, (1995)
12. Reinhart, T.: Pragmatics and linguistics: An analysis of sentence topics. Philosophica **27** (1981)53-94
13. Rich, C., Sidner, C. L.: COLLAGEN: When agents collaborate with people. In: Huhns, M.N. , M. P. Singh (eds.): Readings in Agents. Morgan Kaufmann, San Francisco (1998) 117-124
14. Singh, M.P.: Agent communication languages: Rethinking the Principles. IEEE Computer **31** (1998), 40-49
15. Smith, I. A., Cohen, P.R., Bradshaw, J. M., Greaves, M., Holmback, H.: Designing conversation policies using joint intention theory. In Proceedings of the Third International Conference on Multi-agent Systems. IEEE Press, (1998) 269-276
16. Winograd, T., Flores, F.: Understanding computers and cognition. Ablex Press, New Jersey (1987)