Elio, R., Haddadi, A., & Singh, A. (2000). Task models, intentions, and agent communication. *Lecture Notes in Artificial Intelligence 1886: Proceedings of the Pacific Rim Conference on AI (PRICAI-2000)*, 394-403. Springer Verlag: Berlin

Task Models, Intentions, and Agent Conversation Policies

Renée Elio[1] Afsaneh Haddadi[2] Ajit Singh[1]

[1]Department of Computing Science, University of Alberta,
Edmonton, Alberta Canada, T6G 2H1
ree@cs.ualberta.ca ajit@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. Lengthy message exchanges require some infrastructure to lend them "global coherence." 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 do simple run-time intention recognition without creating complex multi-state protocols. In this approach we promote developing abstract task models and designing conversation policies in terms of such models. An implemented agent assistant based on these ideas is briefly described.

1 Introduction

Recently, there has been considerable interest in specifying agent *conversation policies* [4], which speak to a range of matters in managing lengthy message exchange, from turn-taking and message time-out conventions to responding to dynamic constraints imposed by the environment. Our concern here is with what some researchers [3] claim is a crucial function of a broadly-defined conversation policy, namely constraining "the messages that appear on the wire." It is argued that this need arises from the many-to-many mapping between an agent's intention 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 [5,7]. 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 [5,6,11,13].

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. There is no consensus here either on what the primitive protocols are, let alone their semantics [3]. 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. When a dialogue expands beyond 2-3 message sequences, there must be some way to ensure *global coherence*, i.e., a coherence to how short message sequences are, crudely put, patched together.

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 what we call an abstract task model. 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. This paper is primarily an explication of this position and its pragmatic import. We have realized the ideas presented here in an implementation of a simple agent assistant using a BDI architecture [12].

2 Abstract Task Types and Intentions

For us, an abstract task type is something like "scheduling," "negotiation", "database search", or "diagnosis." Similar notions of generic tasks have supported domain-independent methodologies and architectures for developing knowledge-based problem-solving systems [1]. It seems useful to explore the notion that two agents begin a communication exchange with ‘knowing’ that their joint (abstract) task is one of negotiation, diagnosis, database search, or whatever. To do this, the agents must explicitly share an ontology for the abstract task they are jointly solving, and this ontology is different from the ontology for the actual domain (e.g., medical diagnosis vs. fault diagnosis). It is this meta-level ontology for the abstract task that defines initial, intermediate, and solution states and also defines how movement through those states can be accomplished. Without such a model, it seems that two agents cannot recognize when to begin exchanging messages, what protocol to initiate next, whether progress on the task is being made, or even when message exchange can stop.

We adopt a pragmatic perspective of intention as a commitment to goal, with a specification of when and how the goal is to be pursued and when the goal is abandoned [6]. Intentions move an agent to act and in multi-agent systems, they can be the impetus to engage in dialogue with a collaborating agent in order to act. Thus, intentions drive the conversation and it is these intentions that make multi-message sequences about some task globally coherent.

An abstract task model supports coherence beyond a single protocol by defining intentions as goals to achieve, a general strategy for goal-ordering, and methods for achieving those goals. By defining intentions, an abstract task model also serves to specify and constrain the content of individual messages by defining (i) what the messages can be about, or what we call message objects or "objects of discourse," (ii) what legal intentions (computational actions) an agent can have towards those objects, and (iii) how to progress on the task and when the task is completed. We address what we call local coherence as many others have, namely by specifying protocols for conversational sub-units. Unlike most protocols, our protocols are restricted to message-type *pairs* whose definition includes an option for a "violation" or "unexpected message." While a message type may be "unexpected" in the context of some protocol, its underlying intention cannot be unexpected in the context of the abstract task model that is jointly held by the two agents. Thus, an agent must be designed with (i) the notion that protocols *can* be violated, (ii) a mechanism for recognizing the intended new context, and (iii) the ability to evaluate the implication of the new context for the resumption of the current protocol (and its associated intention). That these notions are crucial to multi-agent cooperation has been argued from a theoretical perspective (e.g., [2]), but solutions implied by that work require systems with sophisticated reasoning engines. Our implementation illustrates that these can be handled flexibly and pragmatically with limited "reasoning" enabled by an abstract task model methodology.

3 Motivating Application: Successive Refinement Search

Our motivating application called for an agent that would assist a human (e.g., a driver in an automobile) in a variety of tasks, which include location and retrieval of information from various databases. The database entities are defined by feature-value tuples. This application is sufficiently challenging under the assumptions that: (a) the database is so large that the human agent cannot specify a single, simple database query for which there is one single answer; (b) the human agent may not necessarily have a definitive set of constraints in mind when starting the search task, or may change constraints during the message exchange, (c) there may be multiple search goals, and (d) the interface cannot be visual or require direct manipulation. Although we present our distinctions terms of human-agent communication, the general issues of intention recognition, resolving unexpected messages, and shifting initiative are generally relevant to agent-agent communication as well. In the following example dialogue (with English gloss), DBA designates the database agent and U designates the user.

U-2: I want inexpensive food in San Francisco.

DBA-3:	O.K.. There are several hundred inexpensive restaurants in San Francisco.
DBA-4:	What type of food do you want?
U-5:	What do you mean?
DB-6:	I mean ethnic type such as Hungarian or French.
U-7:	Italian.
DBA-8:	O.K. Do you want a take out or sit-down place?
U-9:	What about on the Peninsula?
DBA-10:	O.K.-- we'll leave San Francisco for a moment.
DBA-11:	There are 35 Italian restaurants on the Peninsula.

Our concern here is the analysis of the task at a level of abstraction and generality that supports the specification of a jointly held abstract task model. We include the above dialogue for illustrative purposes only: the task model is not derived from dialogue analysis (which might serve as a starting point) but from task analysis. 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 problem 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. The message objects (which define different task state) are (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) s*earch-space members*: particular entities within a particular search space. What can be said about these objects is restricted to the computational actions that can be taken with them, and these are the intentions an agent can have about them. 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, (iii) *expanding* that search space by relaxing one or more constraints, and (iv) *describing* information about a particular member of the search space, about a particular constraint, or about the search space as a set. These traditional database operations may be augmented with other capabilities that are unique to each agent (e.g., an agent might also compute the most-discriminating feature for a given search space). Generally speaking, these operators are the *only* that are legal actions for an agent. 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*. The satisfaction of these intentions causes a movement to another state in the space, though some computation that changes the problem. 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 objects that comprise task states.

In the above message exchange, U-9 does not provide an answer to the question posed in message DBA-8, and instead shifts the direction of the task. Handling this case could be done with protocols with additional transition arcs. But taken to the extreme, this solution must anticipate every such possible adjacency and represent those possible adjacencies with conditional arcs. Defining semantics for a completed protocol becomes difficult, since many alternative paths might be traversed before the protocol exit state is reached. As we present below, we use two-state protocols and shift part of the burden away from anticipation (by the designer) and onto recognition (by the agent). *Intentions that an agent can have about the task (and presumably express during the message exchange) 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 set of task intentions.

The abstract task specification may also define *discourse intentions* —the commitment to perform a communication acts. These arise in service of task intentions. The simplest example is when one agent is known to have information necessary that another agent recognizes is necessary to advance the task (i.e., to satisfy a task intention). Thus, task operator preconditions delimit one set of objects for discourse intentions. A second set of discourse intentions is defined by a task operator’s post-conditions, which define a new successor state. An agent must form discourse intentions to communicate these post-conditions, if another agent without direct access to these post-conditions must know about them to fulfill its role in the joint tasks. Conversely, there are tasks in which the post-conditions of a task action ought not to be shared. Such specifications (which participating agent must, or must not, know what) are rightly part of an abstract task model as well. In essence, the abstract task model defines a protocol at the level of intentions that must, may, or may not be held by each agent. This too can be part of a shared conversation policy.

4 Message Types, Objects, and Content

The semantics underlying the language primitives are based on a number of pragmatic principles discussed in [6]. Our primary concern is not with the syntactic form of the message but with specifying its content. We use the schematic format *(performative $agent-name1 $agent-name2 $object-of-discourse $content)*, although any ACL message syntax with additional necessary parameters might be employed.

The outermost performative, or message type, represents the general class of a message. In ATS, we make use of the classes *request*, *query*, and *inform* (see Table 1). 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. We discuss *$object-of-discourse*, which is constrained for each performative class, below. The *$conten*t of a message can be another ("inner") performative, which further specializes 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 second 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.

Table 1: Performatives and their combination

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

Request. Following [6], we view a request as having an associated level of commitment. A request performative is tightly coupled with advancing the 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. When a request is made by the database agent, the agent is making a pre-commitment to how the progress on the search task might be accomplished and it prompts the user agent for information in order to do this. Requests from the database agent 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. By supplying the information asked for, the user is committing to this computation. When a request is made by the user agent, 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. The objects-of-discourse that may accompany suggest and provide are (i) the *domain* (e.g., restaurants, hospitals); (ii) the *value* of one or more specified features that comprise a constraint; and (iii) the search space *(i.e., the set of database entities described by the current set of constraints).* Table 2 provides some examples of agent and database agent requests (with English gloss).

Table 2: Example request and query messages

Req-U: Let's look for a restaurant in Mid-Peninsula.
(request U DBA :action (initiate :task search :domain restaurants))
(request U DBA :constraint (provide DBA U :value (fv-pairs :feature location :value Mid-peninsula)))
Req-DBA: How about French?
(request DBA U :constraint (suggest DBA U :value (fv-pairs :feature rest-type :value French)))
Que-U: What do you mean by restaurant type?
(query U DBA :knowl-base (provide DBA U :domain (describe-feature :domain $domain-id feature-list :feature rest-type :attribute range))
Que-U: What are the opening hours for this restaurant?
(query U DBA :database (provide DBA U :member (describe-member :member $member :feature hours)))

Query. A query is not about advancing a task but about exchanging information. Its objects of discourse are (i) agent *capabilities,* (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 database agent must respond with an *inform* followed by an appropriate inner performative. The objects-of-discourse that may accompany the inner performatives associated with queries can be (i) the *domain* (e.g., restaurants, hospitals); (ii) the current search *space*; and (iii) a particular *member* in the current search space or database.

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.

The crucial aspect of this analysis is not its realization in these particular message objects. Rather, it is that message objects and content are defined by some commitment to an abstract task model, which can be part of a conversation policy.

5 Protocols as Operators

A task intention concerns advancement of a task-related goal or sub-goal and a discourse intention concerns advancement of information exchange, in support of a task intention. We believe it is important to model the task intentions and discourse intentions as characterizing two distinct state spaces. Whenever an agent cannot proceed in the task space, it adopts a discourse intention (e.g., sends a query message to gather information). This transfers control to discourse space until sufficient information has been gathered from the other agent to proceed in the task space. Similarly, whenever an agent cannot proceed further in the discourse space (e.g., answer a received query), a task intention is formed to compute the necessary information (causing a state transition in task space) in order to satisfy a discourse intention and allow a transition to a new state of shared information.

In our view, *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 operators for computing task successor states. 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. Discourse protocols are similarly defined, while additionally representing the temporal and contextual conditions on what message performatives may follow each other, as per Table 1.

This discourse vs. task space distinction has, we believe, both theoretical and practical import agent design. It supports a clean separation of semantics associated with agent knowledge (modeled in discourse space as satisfied discourse intentions) from semantics associated with progress on the task (modeled as progress in task space through satisfied task intentions).

6 Implementation of a Database Assistant

We have built the agent-assistant for successive refinement search based on abstract task model and the distinctions outlined above using PRS-CL [10], a BDI style architecture. We have put forward our groundwork for these distinctions, at the expense of having the opportunity to provide many implementation details. Here, we emphasize 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.

Generally put, a BDI architecture consists of a dynamic (working) memory that holds current goals and beliefs (symbolic patterns), a plan 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. Due to space constraints, we assume the reader is familiar with the execution cycle of these architectures [12].

Within this architecture, we functionally partition dynamic memory into a task space and discourse space. Task and discourse intentions, as well as task and discourse protocols for message exchange, are implemented as different plan types the plan library, and their invocation and resumption is influenced by a priority scheme defined for those types. 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. Plans that represent task intentions either trigger discourse intentions (if they cannot be immediately satisfied) or pass control to task protocols (if their satisfaction conditions are satisfied and information can be sent to the foreign agent for some computation). Similarly, discourse intentions may give rise to task intentions or pass control to discourse protocols to initiate message exchange with the user. The plan bodies for all discourse protocols 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 plan determines 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 check whether all its satisfaction conditions are met. If not, its plan body may set another discourse intention.

If an incoming message is not recognized as an expected message by a waiting protocol, the message handler releases it to dynamic memory. 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: a plan from the plan library is triggered.

Our database agent handles the conversation flow illustrated by the sample dialogue, as well as others in which there are several changes of direction, multiple sub dialogues embedded within sub-dialogues that in turn shift the task direction, and so forth. This application domain is admittedly simple: there are 24 intentions in the plan library corresponding to task and discourse intentions, and discourse protocols.

7 Related Work and Discussion

Several researchers have specified semantics at protocol-level, realized as pre-, post- and completion conditions [2, 7, 13]. 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. We have argued for a clean separation between discourse and task spaces, believing that it clarifies what the "local" and "extended" effects of a successfully-completed message exchange are [9]. For us, local effects of completing a message exchange via a protocol are effects on the discourse space. 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 [13, 14]. By our model and in our implementation, these are post conditions associated with satisfied discourse intentions (plans), that were accomplished via completed discourse protocols (also plans). The extended perlocutionary effects of a successfully executed discourse protocol, in our framework, are specified as changes first to the discourse space and then possibly, through an associated task intention, as changes to the task space.

Pitt and Mamdani [9] 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 have pragmatically realized many aspects of their protocol-based semantics in our current implementation via the abstract task specification, which designates the relationship between task intentions, discourse intentions, and ultimately, well-structured communication acts.

We are considering a suitable representation for presenting an abstract task specification as a downloadable conversation policy. We particularly need to examine the matter of agent roles and functionality within the abstract task specification. Further, in our implementation, there were numerous control difficulties that resulted from representing both intentions and protocols as plans, allowing reactive recognition of new intentions where appropriate, and balancing sequential and reactive elements of the system. We are considering architectures like SOAR [8], in which an "impasse" in one problem space spawns a new problem space with associated operators. This architecture would support a clean separation of task and discourse semantics (as different problem solving spaces), but at the expense of a plan-based approach to intention representation. 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 one step towards coherent message exchange, based on "intentional states" that can be directly traced to a representation of the global task that may be part of a public conversation policy specification.

Acknowledgements

This work was supported in part by NSERC research grant A0089 to R. Elio.

References

1. Bylander, T., Chandrasekaren, B.: Generic tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge engineering. Int. J. Man-Machine Studies **26** (1987) 231-243
2. Cohen, P. R., Levesque, H. J.: Communicative actions for artificial agents. In J. M. Bradshaw (ed.), Software Agents, AAAI Press, Menlo Park CA, (1997) 419-436
3. Greaves, M., Holmback, H., 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

4. Greaves, M, Bradshaw, J. (eds.): Specifying and Implementing Conversation Policies, Autonomous Agents '99 Workshop, (Seattle, WA, May 1999)
5. FIPA-99 specifications, see www.fipa.org/spec
6. Haddadi, A.: Communication and cooperation in agent systems: A pragmatic theory. Lecture Notes in Computer Science, Vo1.1056 Springer-Verlag, Berlin 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. Laird, J.E., Newell, A., and Rosenbloom, P.S. SOAR: An architecture for general intelligence. Artificial Intelligence **33** (1987) 1-64
9. 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
10. Myers, K.: A procedural approach to task-level control knowledge, Proc. Third Int. Conf. on AI Planning Systems, AAAI Press, Menlo Park CA, (1996), 166-173
11. 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
12. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. Tech. Rep. 56, Australian Artificial Intelligence Institute, Melbourne, Australia, (1995)
13. Smith, I. A., Cohen, P.R., Bradshaw, J. M., Greaves, M.,Holmback, H: Designing conversation policies using joint intention theory. In Proc. Third Int. Conf. on Multi-agent Systems, (Paris, France 1998) IEEE Press, 269-276
14. Sidner, C. L.: An artificial discourse language for collaborative negotiation, Proc. of 12th. Int. Conf. on Art. Intell (AAAI), AAAI Press, Menlo Park CA, (1994) 814-819.