Petrinjak, A., & Elio, R. (2003). Understanding "not-understood": Towards an ontology of error conditions for agent communication. In Y. Xiang and B. Chaib-draa (Eds.) *Advances in Artificial Intelligence: 16th Conference of the Canadian Society for Computational Studies of Intelligence, LNAI 2671*, Halifax, Canada, June 11-13. pp. 383 - 399. Springer-Verlag, Berlin.

Understanding "Not-Understood": Towards an Ontology of Error Conditions for Agent Communication

Anita Petrinjak and Renée Elio

Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1
{anita,ree}@cs.ualberta.ca

Abstract. This paper presents the notion of an agent interaction model, from which error conditions for agent communication can be defined—cases in which an agent generates a *not-understood* message. Such a model specifies task and agent interdependencies, agent roles, and predicate properties at a domain-independent level of abstraction. It also defines which agent beliefs may be updated, revised, or accessed through a communication act from another agent in a particular role. An agent generates a *not-understood* message when it fails to explain elements of a received message in terms of this underlying interaction model. The reason included as content for the *not-understood* message is the specific model violation. As such, *not-understood* messages constitute a kind of 'run-time error' that signals mismatches between agents' respective belief states, in terms of the general interaction model that defines legal and pragmatic communication actions. The interaction model can also set policies for belief revision as a response to a *not-understood* message, which may be necessary when task allocation or coordination relationships change during run time.

1 Introduction

One cornerstone of the software agent paradigm has been the effort to define and standardize a high-level language for agent communication. The Federation for Intelligent Physical Agents (FIPA) offers one such standard for an agent communication language (ACL) [9]. This standard defines core message types (e.g., *inform*, *request*, and *query*) with associated semantics based on an underlying theory of communication as rational action [3, 20]. To be compliant with this standard, an agent need not be able to process all the predefined core messages. But there is a *single* core message that all agents must be able to generate and interpret. This message is *not-understood.* The form of this message is *(not-understood :sender j :receiver i :content c),* where *i* and *j* are agents. The *not-understood* message is *j*'s response to *i* in the context of some previous message from *i* to *j*. The message content *c* is defined as a tuple consisting of an action or event (e.g., a just-received *inform* message from *i* to *j*) and a 'reason.' The occasion for sending *not-understood* is described as follows:

> The sender of the not-understood communicative act has received a communication act it did not understand. There may be several reasons. [An agent] may not have been designed to process a certain act or class of acts, or it may have been expecting a different message. For example, it may have been strictly following a predefined protocol, in which the possible message sequences are predetermined. The not-understood message indicates to the receiver that nothing has been done as a result of the message....The second term of the [content] tuple is a proposition representing the reason for the failure to understand. There is no guarantee that the reason is represented in a way that the receiving agent will understand. However, a cooperative agent will attempt to explain the misunderstanding constructively. [9]

The *not-understood* message is pragmatically quite important for agent communication. The basic premise of speech act theory [1,21], and its adoption in computational accounts of communication as planning [3,4] is that communication acts are actions upon a world. By FIPA's formal semantics, if agent *i* informs agent *j* of some proposition ϕ, agent *i*'s intended effect is that *j* adopt belief in ϕ. Agent *j* may not do so for a number of reasons, and hence the intended effect is just that—intended but not guaranteed. Thus, in making a communication act, a speaker is aiming to change an aspect of a non-deterministic world that is not directly accessible, namely the mental state of the hearer or receiving agent. Hence, *not-understood* is a quite pointed response back to agent *i* from that inaccessible world, indicating that the action was not (merely) unsuccessful but, to put it crudely, dead on arrival for some other reason. Our interest here is in defining those conditions that must be met before *j* would even attempt to assimilate ϕ into its informational state.

The work we describe here proposes the notion of an interaction model for defining and structuring reasons for generating a *not-understood* message type, and also for defining possible responses to receiving a *not-understood* message. Our approach is to use an interaction model to define expected, legal, and pragmatic messages in ways that are more general than predefined protocols. Deviations from this interaction model are occasions for generating a not-understood. An ontology of error conditions for agent communication flows directly from an interaction model and the different ways in which messages might constitute deviations from this model.

There are several reasons why we think it is theoretically and pragmatically useful to take a serious look at *not-understood*. First, thinking about *not-understood* is a different way of thinking about what it means to *understand*. We can think of understanding as the message's extended perlocutionary effects, i.e., how that message affects an agent's internal state, such that it behaves differently for having received and assimilated the message [see 8, 18]. Second, the considerable effort in specifying the syntax and semantics for agent communication languages does not address the matter of what agents actually communicate *about*, i.e., what fills the *:content* field of a message, even when there is a shared ontology and a shared content language. It seems that, for some applications, what cooperating or coordinating agents 'talk about' is their problem-solving progress. The abstract specification of task interdependencies

and what constitutes task progress can serve as part of a jointly held conversation policy [8, 10]. Finally, if we seriously regard the notion of agent as a kind of programming abstraction [22], then *not-understood* can be viewed as signaling a 'run-time error.' The analogy is this: a communication action generates an error when its effect would constitute an illegal or impossible action on the receiving agent's internal (mental) state. The matter at hand, then, is to define that set of illegal or impossible actions, and for that some kind of model is needed.

Generally speaking, our view is that an interaction model defines what agents can, must, or might talk about during their joint problem solving, and how this may be further constrained through the specification of agent roles. A *not-understood* is the detection of a discrepancy between the messages that are allowed by that model and what messages are encountered. The 'constructive reason' used in the *:content* field is a specific type of violation, stated at the same abstract level as the interaction model. Now, if agent *i* receives a *not-understood* concerning one of its messages, it would be good if agent *i*'s internal state changed, so that it did not send that very same message again. The inclusion of a 'constructive reason' that is interpretable by agent *i* opens the possibility for belief revision, and this is a complicated matter. We discuss our initial ideas about how an interaction model might set some policies for this as well.

2 Occasions for not-understanding an inform

Our current interaction model has three main elements: (a) the ontology of predicates that are interpretable by the agents, (b) what kinds of propositions using those predicates can be used in the *:content* field of particular message types (i.e., the object of particular illocutionary forces); and (c) which agents are permitted to send particular message types, with particular content, to which other agents. Together, (a) and (b) define what kinds of beliefs in agent *j*'s mental state could be updated, revised, or accessed through an external communication action from agent *i*. Element (c) further constrains these operations to be legal only when agent *i* holds a particular role to agent *j,* in the context of some cooperative or coordinated behavior.

To illustrate the general intuitions behind our approach, the top portion of Table 1 presents a schema for *inform*, which takes a proposition φ as its content. The schema includes FIPA's feasibility preconditions and rational effects for *inform*. We also include contextual relevance conditions, i.e., that the receiver of the *inform* wishes to know φ [3, 13, 21]. Finally, we specify certain success conditions [21] that stipulate (some) conditions that must hold for the *inform* action to be successful. There are six conditions in total for our *inform* schema. An *inform* is legal to send—and understandable to receive— when these conditions are holding. Conversely, an *inform* is *not-understood* if, from the receiver's perspective, one or more of these conditions ought not to be holding in the sender's internal state. This is what the lower part of Table 1 shows.

Specifically, the lower portion of Table 1 illustrates six types of *not-understood* that are defined by the six conditions in the upper part of the table. The first three are

concerned with the propositional content of the message itself, the particular illocutionary force applied to the content, and the role of the sender to the receiver. Case (i)

Table 1. An *inform* schema that defines six cases for *not-understood*

<*inform i, j, □*>	
success conditions:	i. *□* is interpretable by *j*
	ii. *j*'s state concerning *□* can be updated
	iii. *j*'s state concerning *□* can be updated by *i*
feasibility preconditions:	iv. *i* knows *□*
	v. *i* believes *j* has no position on *□*
contextual relevance conditions:	vi. *i* believes *j* desires to know *□*
intended effect	*j* adopts belief in *□*

<*inform i, j, □*> is not-understandable to *j* wrt an interaction model *m* if by that model

i) *□*'s predicate is not in a commonly shared ontology
- the intuition: "I don't know what *□* means."
- model component: public vs. private predicates

ii) *j*'s position on □ cannot be updated/revised
- the intuition: "My belief about *□* cannot be changed."
- model component: static vs. defeasible predicates

iii) *j*'s position on *□* cannot be updated/revised by a communication act from *i*
- the intuition: "*You* cannot change my belief about *□.*"
- model component: external vs. internally defeasible predicates; agent roles derived from shared task model

iv) *j* cannot explain why *i* would know *□*
- the intuition: "How is it that *you* know *□*?"
- model component: agent models derived from shared task model

v) *j* cannot explain why *i* believes *j* has no position on *□*.
- the intuition: "Why do you believe I do not *already* know *□*?"
- model component: agent models from shared task model and run-time updating

vi) *j* does not desire to know *□*
- the intuition: "*Why* are you telling me *□*? It would have no impact on my behavior."
- model component: agent models derived from shared task model

is a simple matter of whether the predicate of proposition *□* is in the shared ontology of the agents. Such predicates are classed as *public* and are allowable in the propositional content of a message. The interaction model designates *private* predicates as those used to construct belief state propositions that might be idiosyncratic to particular agents, or that ought not to be exchanged. Case (ii) stipulates what propositions that involve public predicates are defeasible, and which are not, by classifying public predicates as either *static* or *defeasible*. Case (iii) covers which defeasible beliefs are

changed through *internal* reasoning actions only or through an *external* communicative act of another agent. For example, an interaction model could specify that agent *i* may not *inform* agent *j* what agent *j* believes, intends or desires, to embody the constraint that revisions to *j*'s mental attitudes are the province of agent *j*. (These cases loosely correspond to canonical examples such as "I insult you." "I convince you of □." [1]). But this distinction can apply more widely to other propositions in *j*'s belief state, namely any proposition for which only *j* can determine a truth status.

Cases (iv) through (vi) focus on *not-understanding* as violations of agent models—beliefs about other agents, their capabilities, their responsibilities, their realm of knowledge, and so forth. Systems of coordinating or cooperating agents often implicitly or explicitly rely on such acquaintance models, derived from a global or partial model of interdependencies among tasks and the agents assigned to those tasks [6,11, 14]. In these cases, a *not-understood* message signals a mismatch between agent *i* and *j*'s respective models of each other. For example, cases (iv) and (v) correspond to *j*'s inability to explain why—given its beliefs about agent *i* and what agent *i* ought to know about agent *j*— the feasibility conditions for *inform* are holding for *i*. Finally, case (vi) covers an important pragmatic case, from the viewpoint of a message's extended perlocutionary effects. Presumably, *i* intends that *j* adopt belief □ so that *j*'s behavior will change. If belief in □ would not impact any of *j*'s behaviors, there is no consequence of adopting it. In this instance, *j*'s *not-understood* signals a mismatch between *j*'s own model of contextual relevance, and agent *i*'s model of what is contextually relevant to *j*. In some situations, it might be quite important for agent *i* to learn that the *inform* it sent to agent *j* would have no impact on anything that agent *j* does. Agent *j* might not coordinate its activities with *i* any differently, release resources any differently, and so on, if *j* cannot understand the *inform* message in terms of its task or agent models. This could result in a domain-level error situation, brought about because an *inform* action did not have its intended effect.

We wish to make a few additional points at this juncture. First, we have not explicitly included the case of protocol violation, which is the typical situation used to motivate or define a *not-understood* scenario. A predefined protocol can be viewed as a special instance of case (vi). We readily acknowledge that protocol violations are more easily recognized as syntactic violations, but that is a matter of processing convenience. Second, we use the term 'explain' in Table 1 to emphasize that a received message must be consistent with the receiver's model. We have no particular investment in how simple or complex this consistency-checking process might be nor do we expect that agents will necessarily discover a discrepancy by means of some theorem proving procedure applied to formulas in the semantic language that represents their mental states. Finally, it is natural at this juncture to speculate whether agent *i*, having received a *not-understood* message from agent *j*, would itself generate a *not-understood* message back to agent *j*. And on *infinitum*. This can be a difficult issue. Our initial approach for avoiding this is to appeal again to the interaction model, and we address this in a later section.

We have focused to date on FIPA message types that take propositions as message content, namely *inform* and *disconfirm*. Our treatment also handles a variation of *query* (namely, *query-ref*), whose semantics under the FIPA specification are (com-

posed of) a *request* for an *inform* action. FIPA includes *refuse* as a possible response to a request for an action, although a *query* might generate a *not-understood* by our analysis in Table 1. For brevity's sake, we limit our discussions and examples to *inform* in the remainder of this paper.

3 Elements of An Interaction Model

One purpose of an interaction model is to define the space of possible intentions that cooperating or coordinating agents can have. A second purpose is to define the occasions on which it is expected or likely that those intentions would arise. A third purpose, related to these first two, is to delimit message content, i.e., what appears in the *:content* field of a particular message.

There are three main components to defining an interaction model that would support the generation of *not-understood* messages illustrated in Table 1. The first is a language to describe tasks and interdependencies among tasks, and hence interdependencies among agents assigned to those tasks. TÆMS [7] is an example of such a language and we adopted several of its distinctions. The second is a set of axioms and inference rules that agents can use, together with a model of task dependencies, to derive initial and run-time propositional beliefs about other agents and other tasks. It is these propositions that agents will exchange, update, or revise through communication actions during a problem solving episode. Particular exchanges, revisions, and updates are then understood as required, expected or plausible, *given* these task and agent models. The third component concerns definitions of particular predicate classes, that define what types of message content can be exchanged under various circumstances (e.g., the static vs. defeasible, internal vs. external distinctions described in Section 2). The remainder of this section aims to present just enough detail about our current interaction model framework, to ground our examples from Section 2 and our later discussion about replies to receiving a *not-understood*.

3.1 Task Structures

We model a task as an abstract specification of a problem to be solved that has a number of defining properties. Following [7], the solution method for a task is specified as either a directly executable *method* or via the achievement of a set of *subtasks*. A (sub)task is related to another task through either an *and*-decomposition or an *or*-decomposition. Tasks may also be related to each other via an enables/enabled-by relationship: if task *i* enables task *j*, then task *i* must be completed before task *j* can begin. A *task-structure* specifies a hierarchical decomposition of a task into a set of subtasks whose leaf nodes are executable methods. In our modeling assumptions, each task and method in a task structure is assigned to exactly one agent, although a given agent may be responsible for more than one task or method. Figure 1 shows a fragment of an (abstract) task structure to illustrate these properties.

A task definition also includes pre and post conditions, which are used to define its initial state and its goal state. Pre and post conditions are stated as Boolean constraints on domain-specific variables relevant to initiating a task and deeming it successful, respectively. In our airline ticket reservation application, a simple precondition to the task of obtaining user information would include that the departure and return dates are unknown; a post condition is that both such dates are known and that the former occurs before the latter. Any kind of domain-dependent check can be specified in this way; we assume that agents have the domain-dependent code to determine if the constraint is holding. In our more abstract domain-independent language, the preconditions are regarded as resources that an agent requires to begin work on a task. Similarly, to declare that a task is completed and successful, an agent must acquire beliefs that a task's results have been achieved (i.e., that the post condition constraints are holding). That brings us to the matter of abstract task operators.

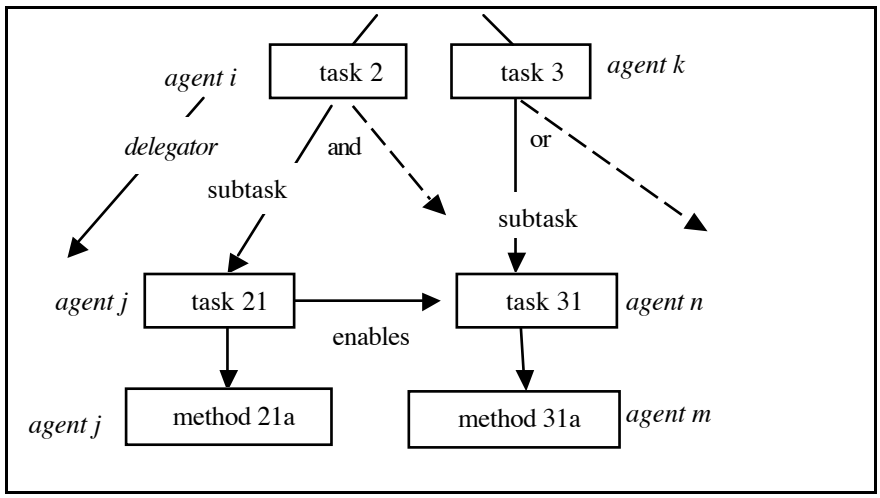

Fig. 1: Elements of task structure and derived agent roles

3.2 Task Progress and Agent Beliefs

Our interaction model also uses a domain-independent vocabulary for task progress. A task can be in one of these states: *not-attempted, possible, not-possible, irrelevant, failed,* or *succeeded*. An agent is designed to move its task from its initial state (*not-attempted*) to a final state (*irrelevant, failed*, or *succeeded*). It does so by applying what we call abstract task operators. An example abstract task operator, using English gloss, is:

If I intend task *t*, and status of task *t* is currently not-attempted
& I have all the resources and information specified for task *t's* initial state
& all semantic constraints on those preconditions are satisfied
then change task *t*'s status to possible

This operator essentially captures the notion that problem solving on a task can commence, i.e., the constraints that constitute its preconditions have been met and therefore progress on the task can commence. Application-specific code is used to instantiate the preconditions in these operators. These operators thus serve to bridge task dependent constraints with a task independent ontology that agents can use in their communication acts.

All agents have access to the entire task structure (the complete decomposition of a root task and the assignment of subtasks and methods to other agents), although this is not strictly necessary for our purposes. An agent's belief state is initialized and modified during run time by the agent's use of the shared task structure coupled with a set of axioms and inference rules. This is the second, major component of the interaction model, which serves to unite a model of task interdependencies with a traditional belief-desire-intention perspective for modeling an agent's internal state. Some examples of these rules (stated in English) are: "*Every task has an assigned agent and only one such agent*." "*Only the agent assigned to a task can intend it*". "*An agent desires resources for its assigned task iff the agent intends it*." The further explication of our axioms and inference rules is outside the scope of this short paper and not central to our concern. However, the general character of these rules is that they define and constrain task properties on the one hand and agent properties (e.g., roles, attitudinal states) on the other. The role of agent *i* to agent *j* in Figure 1 as *delegator* is derived from such inference rules applied to the jointly held task model. Similarly, the belief that agent *n* desires the results from task-21 is derived from beliefs that agent *n* is the agent assigned to that task, and that task-21 enables agent *n*'s task. These ultimately serve as the states that correspond to preconditions for communication acts. Table 2 shows a subset of the beliefs that would follow from these sorts of inference rules, coupled with the task structure in Figure 1.

Table 2. Illustrative belief state elements for agent j, Fig. 1

beliefs about self	(agent-for j task-21) (relevant task-21) (intend j task-21) (agent-for j method-21a) (desire (j (result-for (task-21 result-a))) (enables task-21 task-31) (subtask-of task-21 task-2) ...
beliefs about others	(agent-for n task-31) (agent-for i task-2) (intend n task-31) (relevant task-31) (desire (n (result-of (task-21 result-b)))))....
beliefs about tasks	(relevant task-21) (status task-21 possible) (have-all-resources task-21) (valid-all-resources task-21) (decomposition task-21 method-21) ... (status method-21 not-attempted) ....

At the implementation level, we allow one level of nesting for belief propositions, and drop the outer most *believe*. *(Desire agent* Φ*)* is used to signify that an agent will aim to achieve a belief state in which proposition Φ is true. The *intend* predicate sig-

nifies an attitude towards a task by an agent; *intend (agent-i task-23*) is shorthand for signifying a behavioral commitment on agent *i*'s part to bring task-23 to a final state. In using this kind of vocabulary, it is important to specify axioms to define the semantics of these concepts. From our viewpoint, domain-specific computations on domain-specific information states can be carried out in any fashion, as long as they ultimately create beliefs in the agent that are represented using this (or some other) abstraction vocabulary. The implementation must ensure that, during execution, agent belief states are internally consistent with the axioms defining the semantics.

3.2 Communication plans

Within each agent, a communication plan module handles the generation and assimilation of *inform*, *disconfirm* or *query* messages. Communication plans are triggered in two ways. First, an agent may select a particular communication plan as the means for satisfying the preconditions of an abstract task operator. For example, an agent may execute a *query* plan to receive information about resources that are necessary to move its task from *not-attempted* to *possible*. Second, an agent may proactively generate an *inform* or a *disconfirm* message to exchange information it believes will be useful to other agents. For example, an agent that determines its own task has failed may immediately inform other agents who require its results or resources. The communication plans implement the kind of schema illustrated in the top panel of Table 1. An agent's communication module also creates the necessary data structures for maintaining conversations between itself and several agents. Incoming messages are recognized as either completing or continuing an on-going conversation or as initiating a new conversation, provided that such messages pass the *not-understood* filters (described below).

3.4 Declarative and Procedural Forms of the Interaction Model

In our framework, agents share a declarative form of the interaction model represented in XML, with document type definitions for the key components that follow. The first component is the definition of a task structure using the general task description language, with agents assigned to specific tasks and methods within this structure. The second component is the set of allowable message types. In our sample model, this set has the members *inform, disconfirm, query-ref,* and *not-understood.* The third component is the set of allowable predicates that can comprise a proposition in an agent's belief set. Each of these predicates must be classified along three dimensions, defined generally in Section 2: public or private, static or defeasible, internal or external. Only propositions that involve public predicates may appear in any message content. Propositions involving defeasible predicates may be revised during run-time; those involving static predicates may not. External predicates may appear in the propositional content of inform or disconfirm messages; internal predicates may not. The fourth component is a specification of agent roles. In our current interaction model, there are two types of agent roles which are isomorphic to task dependencies: delega-

tor—delegatee (isomorphic to task—subtask) and enabler-enablee (isomorphic to task dependencies of enables—enabled-by). However, an interaction model could include agent roles that are not isomorphic to task structure. The fifth component of the interaction model defines what combinations of message type, agent-role, and predicate type are allowable, i.e., which illocutionary forces can be applied to particular public, external, defeasible predicates by an agent having a particular role relative to another agent. Here is a portion of model that captures these last two components:

public: { resource task-status task-relevance intend agent-for...}
defeasible: { resource task-status task-relevance intend ...}
external: { task-relevance resource}
inform [sender-role: delegator] [predicate-class: external] [object: task-of(receiver)]
query [sender-role: any] [predicate-class: public] [object: task-of(receiver)]

This model fragment indicates that agent *j*'s beliefs about the relevance of its task can be updated via an *inform* act from agent *i*, if agent *i* had delegated that task to agent *j*. (This is just one specification for *inform*—the model would specify several of these.) This particular specification also permits any public predicate to be the content of a query, regardless of the respective roles of the sender and receiver. A different specification could constrain queries as a function of agent roles. We have mentioned the importance of axioms and inference rules that unite these abstract interaction elements; these are not in our XML representation, although a more expressive modeling language [e.g. 5] might allow such axioms to be part of a declarative representation.

inform-messages □	inform-1 | inform-2
inform-1 □	(inform sender receiver role-of$_{s\text{-}to\text{-}r}$ (task-subject))
task-subject □	task-of$_{sender}$ | task-of$_{receiver}$
role-of$_{s\text{-}to\text{-}r}$ □	delegator | delegatee | enabler | enablee
external □	relevant | resource
modality □	intend | believe | desire
[delegator] [task-subject] □	[delegator] [task-subject] [external]
[delegatee] [task-of$_{sender}$]□	[delegatee][task-of$_{sender}$][result]
[role-of$_{s\text{-}to\text{-}r}$][task-of$_{sender}$]□	[role-of$_{s\text{-}to\text{-}r}$][task-of$_{sender}$][status]
inform-2 □	(inform sender receiver role$_{s\text{-}to\text{-}r}$ (sender modality)

Fig. 2. A grammar for legal inform messages allowable by an interaction model (*s-to-r* = *sender-to-receiver*)

Upon reflection, it is clear that the interaction model could be used to enumerate the set of all messages that adhere to its constraints. Indeed, the point of the model is to define a finite set of intentions and hence a finite set of messages that could (in principle) be sent between agents. It is convenient and useful to re-represent this functionality as a grammar that an agent can use to generate (or validate) messages. Figure 2 shows such a grammar for generating schematic versions of an *inform* message, created (by hand) from an interaction model specification.

In Figure 2, *Task-of* is a function that takes a particular agent as an argument and returns the task(s) to which that agent is assigned. For convenience, we defined two types of *informs* and the grammar generates schematic templates for each type. An example of an *inform-1* that this grammar generates is:

(inform sender receiver delegator (task-of$_{sender}$ relevant))

This *inform* is consistent with the interaction constraints that it is legal and *pragmatic* for an agent *i* (as sender) to inform agent *j* (as receiver) of agent *i*'s own task relevance, if agent *j* is performing a subtask for agent *i* (i.e., *i* stands as delegator to *j*). Using the Figure 1 task structure, the grammar allows *i* to inform *j* that task-2 is, for example, not relevant. It is legal, because agent *i* can send messages with propositional content about task relevance (relevance is public) and relevance can be updated via communication acts (it is defeasible and external). It is pragmatic, because agent *j* might infer its own task should be initiated (or stopped) by receiving messages about agent *i*'s task relevance. This grammar would not generate the following *inform-1:*

X (inform sender receiver delegatee (task-of$_{receiver}$ relevant))

Using the structure in Figure 1, agent *j* cannot inform agent *i* anything about the relevance of agent *i*'s task (task-2): agent *j*'s role to agent *i* is delegatee, not delegator. Such a communication act is disallowed by the interaction axioms and that is captured in this grammar.

An example of a schematic *inform-2* that this grammar generates is:

(inform sender receiver delegatee (sender believe))

This second type of *inform* captures the notion that a sender can inform a receiver about (only) the sender's attitudes towards propositions. Note that this particular grammar is based on an interaction model that requires that *some* direct role exist between the sender and receiver for such a message in the first place (e.g., the grammar would not generate this message from agent *i* to agent *m* in Fig. 1).

Any agent in our system can run this grammar by first binding *sender* to itself and *receiver* to some particular other agent. The grammar implicitly embodies constraints about public, defeasible, and external predicates *as well as* pragmatic considerations about what constitute plausible *inform*s from one agent to another. The latter are captured through reference to the role that the sender plays to the receiver. We implement grammars like this one for *inform*, *disconfirm* (similar to inform) and *query*. This alternative representation of the interaction model is useful because it gives an agent the procedural capability of (a) generating all legal and pragmatic messages between itself and another agent, and (b) checking whether the general form of an incoming message passes various types of understandability filters. The grammars also serve an important function for us, namely to forbid certain kinds of replies *to* a not-*understood* message. We discuss this in the next section.

4 Not-Understood Responses and Conversations

Having both declarative and procedural specifications for an agent interaction model, it is possible to define a general message assimilation routine. Such a routine parses an incoming message and determines whether a *not-understood* response is warranted, by essentially asking these four questions:

1. Is the predicate in the content proposition a public predicate?
2. Is the illocutionary force applied to this predicate allowed?
3. Does the sender hold the proper role relative to the hearer, in order to apply this illocutionary force to the proposition?
4. Are the feasibility, relevance, and success preconditions (assumed to hold on the speaker's part) consistent with the receiver's interaction model and its current belief state about the sender and the task?

Recall that the general form of *not-understood* is *(not-understood :sender j :receiver i :content (m reason)),* where *m* is a just-received message from *i*. Any of the four checks listed above could generate a *not-understood* with a *reason* that takes one of four possible corresponding forms:

(not-understood :sender :receiver
:content (*m* (not (public <predicate *p* of *m*'s content>)) |
(*m* (not (external <predicate *p* of *m's* content >)) |
(*m* (not (permissible-role sender receiver)) |
(*m* (not □)))

The first three of these cases can be handled by using the grammar. The last case includes a proposition □ that will describes the mismatch between the receiver's model and the sender's model of what feasibility and relevance conditions are thought to be holding. For example, this case might correspond to "It is false that I desire the result of task-23." This could occur if tasks are dynamically reallocated and agent models about task responsibilities are out of sync.

At this point, we could consider the matter of generating and structuring *not-understood* messages as done. This is especially true if we regard *not-understood* messages as true run-time errors, i.e., errors that would bring an agent system to a halt, or at least stop further interaction between two agents.

However, things become more interesting, and complicated, if we try to allow some further resolution of a *not-understood* through additional message exchange. Recall that the reason included with *not-understood* is supposed to be a 'constructive explanation' about the matter. It would be *most* constructive if it caused some change to the internal state of the agent who sent the original message *m*, such that the agent would not simply regenerate message *m* all over again. In this sense, we can interpret the reason in a *not-understood* as an opportunity to bring two disparate models into alignment: the reason constitutes the content of an *inform*.

But this immediately raises the question of whose model is to be taken as true—the sender who generated message *m* or the receiver who asserts it is not understandable? We do not, of course, have a general answer for this, but we believe that the underly-

ing interaction model could be used to represent application-specific policies about this. Consider this possible message exchange scenario:

	message 1	(inform *i j* (intend (*j* ....)))
	message 2	(not-understood *j i* (<message 1> (not (external intend))))
?	message 3	(disconfirm *i j* (not (external intend)))
X	message 3'	(not-understood *i j* (<message 2> (external intend))))

In message 2, *j*'s reason is that *j* believes that *intend* is not an external defeasible predicate, i.e., one whose truth status can be updated by a communication action from another agent. Hence *i*'s message 1 is not allowed under *j*'s interaction model. Now, there are two possible responses that *i* could make. In message 3, *i* aims to revise *j*'s model about whether the *intend* predicate is external. Is message 3 allowable? It depends solely on the underlying interaction model: if the predicate *external* is itself an external, defeasible predicate, then message 3 is allowed. Otherwise, it is not. (This does not settle the more general matter of whose model should be taken as correct, but perhaps that can be stipulated via agent roles that make sense within the realm of a given application). The second possible response is message 3', namely that *i* tells *j* that *j*'s *not-understood* message is not understandable to *i*. Within our framework, this cannot happen, because to do so is tantamount to *i* informing *j* what *j* already knows from message 1 (that *i* believes that *intend* was an external predicate). This would violate a feasibility precondition for *inform* (again regarding the reason as the content of an *inform*) and would not be allowed by the model.

The next message exchange illustrates a case where agent *i*'s model of task and agent interdependencies is incorrect, and *i* informs *j* of some result *j* does not need:

	message 4	(inform *i j* (result-of (task-23 ....)))
	message 5	(not-understood *j i* (<message 4> (not (desire (*j*, result-of (task-23...))))
X	message 6	(disconfirm *i j* (desire (*j*, result-of (task-23)))

According to the interaction model, message 4 is not understood because, from *j*'s viewpoint, the contextual relevance conditions for this *inform* should not be holding for *i*: *j* does not need to know the result of task-23 and *i* apparently believes otherwise. The *not-understood* reason, taken as an *inform* from *j* to *i* about *j*'s beliefs, is legal under the interaction model. This can cause agent *i* to update its model of agent *j*'s responsibilities. But agent *i* cannot try to revise agent *j*'s belief state with message 6, because by the interaction model, *i* cannot change *j*'s state about what *j* desires.

In this last scenario, there could be a different reason why agent *j* does not understand message 4. It might not (according to its own model of agent *i*) believe that an *inform* feasibility precondition holds for agent *i*, e.g., that agent *i* has reason to know the results of task-23. If only agents assigned to a task can know task results and *j* does not believe *i* is the agent assigned to task-23, then it might generate message 5' in response to message 4:

	message 5'	(not-understood *j i* (<message 4> (not (agent-for (*i* task-23)))

? message 6' (disconfirm *i j* (not (agent-for (i task-23)))

Message 6' could be a legal continuation of this exchange, *iff* the interaction model defines *agent-for* to be an external, defeasible predicate and (by the associated axioms) that an agent is the final authority on its task assignments (i.e., it can update beliefs held by others about its task assignments).

We have not considered message exchanges involving *query*, but many of the same issues arise. In the simple scenarios considered here, an agent may reply with a *refuse* to a *query* concerning □ if it cannot resolve, by its underlying interaction model, that the inquiring agent has reason to know □. In this way, an interaction model can be used to enforce certain privacy constraints on information exchanged in a multi-agent system. In general, many issues remain about if and how agents use *not-understood* as a means for aligning disparate belief states. Belief states of cooperating agents could diverge during problem solving, through lost messages or, say, through task reallocation during run time. So it is not unreasonable to consider that this simple belief revision—triggered through communication exchanges—might be necessary for agents to adjust to a changing task environment.

5 Related Work and Themes

There has been general recognition that error conditions need to be specified for agent communication [9]. However, to our knowledge, there has not been work done either in defining such error conditions or in structuring *not-understood* messages through an explicit interaction model for software agents. The framework we present here adopts a number of the pragmatic assumptions that emerge from theoretical notions developed in the discourse understanding community [11, 14, 15, 19]. Explicit representations of tasks and task plans are used in such frameworks to define plausible or expected communication. As we have noted earlier, various approaches to coordinating distributed agents employ rich task environment modeling languages [6, 7] and such languages are central to specifying an interaction model. But the primary focus in that work has not been the motivation and resolution of communication actions *per se*. More recently, there is an effort to link conversation protocols directly to task interaction patterns [23]. Our interaction model requires the specification of axioms that link task interdependencies to agent properties. Such axioms provide the bridge from coordination information to BDI approaches for agent behavior and communication that support to abstract conversations about joint agent goals, envisioned in[23]. Our use of agent roles in specifying interaction conventions is also related to models of social problem solving. Such models describe agents in terms of the actions they are committed to executing, the resources they will need to meet those commitments, and their expectations and beliefs about the actions, commitments, and resource needs of others [2]. [24] uses roles for describing the expectations about individual behavior. Here, we are using agent roles as part of an interaction model for defining the pragmatic as well as legal communication actions that may be taken by one agent upon another agent's internal state.

Our working assumption is that understanding and hence not-understanding can only be resolved through appeal to *some* sort of model. By our view, such a model requires a shared task specification that describes task and agent interdependencies and a set of axioms that relate these specifications to BDI elements for characterizing an agent's internal state. What this means is that a set of cooperating or coordinating agents have a 'deep model' of their joint work and each other. This is easy to achieve in a closed agent system, in which a system designer can impart such models to a set of homogeneous and stable agents. But in such a system, one can argue that there is no real need to adhere to a high level ACL in the first place, because the agents can be programmed to communicate with each other in whatever way the system designer decides is best. A good part of the motivation for a high level standardized ACL was for communication among heterogeneous agents, who most likely have *shallow* models of each other. And that is where we come up against the matter of just what such agents will 'talk about'. There is considerable work on specifying ontologies that can support certain web-motivated types of interactions (e.g., service brokering [16]), but it still seems to us that such agents must subscribe to some kind of underlying interaction model. Fixed conversation protocols can certainly be regarded as a simple interaction model, but —as it is generally recognized — such protocols do not constrain message content, only sequences of message types. Some of the interaction model components we have advocated here (e.g., restrictions on whether particular beliefs can be updated or revised through communication actions taken by agents in particular roles) can add a level of semantic check, even without a deeper shared problem solving model.

In either open or closed agent systems, we think that a case can be made for explicit interaction models of the kind we have considered here. It might seem that providing such models is too much overhead. However, there has been increasing interest in extending and applying software engineering methodologies to the agent paradigm [12, 17, 25] and many of the interaction model components we have advocated here emerge 'for free' in the course of specifying a system design in, say, UML.

6 Conclusions and Future Work

The main contribution of this work is the perspective that messages are *not-understood*, and hence understood, with reference to a shared, declarative interaction model. We have also outlined the kinds of elements that such an interaction model might minimally include. A good portion of agent message types have intended effects that are essentially manipulations, updates, or revisions of the mental (informational) state of the receiving agent. We have considered here how *not-understood* messages can be viewed as replies to illegal instances of such actions on an agent's mental state. The model is the means by which legality and illegality is defined. In our framework, the model includes what feasibility preconditions, success conditions, and relevance conditions ought to be holding for the sender, to take a particular communicative action. We have also included the idea that some kinds of agent beliefs are not revisable by communicative acts, or indeed cannot even be the content of communication

acts. When these constraints are combined with agent roles, the interaction model can become even richer and more complex (e.g., *some* agents get to revise *some* sorts of beliefs held by other agents, depending on their relative roles). The general point is that any and all such elements can be used to define a principled approach to constraining message content, message exchange patterns, and thereby a set of error conditions for agent communication.

We have employed the interaction model in a multi-agent solution to a simple domain task (making airline reservations through different web sites) as our test application. Our system framework instantiates and deploys agents that follow the interaction model as they interact to execute this task. We test the framework's ability to generate and assimilate *not-understood* messages by perturbing the individual agent models and generating messages to other agents that are illegal or non-pragmatic, from their viewpoints. The *not-understood* message exchange scenarios described earlier are handled within our implemented system. As we considered in Section 4, interesting issues emerge when we move beyond the error ontology *per se* and consider *not-understood* messages themselves as opportunities for belief revision during problem-solving. Agent roles are crucial to our analysis, because they so strongly influence the legal and pragmatically expected belief revision and updates that can take place through *informs* and *disconfirms*. If agent task assignment (and roles) are permitted to change during a problem solving episode, there must be consistent and reliable means for synchronizing agents' respective models of each other in the task context. This is an important consideration, since it relaxes the assumption that agents systems are stable and task allocation does not change during coordinated or cooperative problem solving. Our on-going work is aimed at a more careful consideration of using the *not-understood* conversations to synchronize mismatching agent models.

Acknowledgements

This work was supported by an NSERC research grant to R. Elio.

References

1. Austin, J.: How to do Things with Words. Harvard University Press (1962)
2. Bond, A. H.: Commitment: Some DAI Insights from Symbolic Interactionist Society. In: Proceedings of 9th Workshop on Distributed Artificial Intelligence. (1989)
3. Breiter, P., Sadek, M.: A Rational Agent as the Kernal of a Cooperative Spoken Dialogue System. In: Intelligent Agents III (LNAI Vol. 1193). Springer-Verlag (1997) 189-204
4. Cohen, P., Pernault, R.: Elements of a Plan-Based Theory of Speech Acts. Cognitive Science 3 (1979) 177-212
5. DAML Agent Markup Language, www.daml.org
6. Decker, K., Lesser, V.: Designing a Family of Coordination Algorithms. In: Proc. 5th Intl. Conference on Multi-agent Systems. MIT Press (1995) 73-80

7. Decker, K., Lesser, V.: Quantitative modeling of complex computational task environments. In: Proc. of AAAI-93. AAAI Press (1993) 217-224
8. Elio, R., Haddadi, A.: On abstract models and conversation protocols. In F. Dignum and M. Greaves (eds.): Issues in Agent Communication. (LNAI 1916). Springer-Verlag (2000) 301-313
9. FIPA: "Agent Communicative Act Library Specification" available at http://www.fipa.org/specs
10. Greaves, M., Holmback, H., Bradshaw, J.: What is a conversation policy? In F. Dignum and M. Greaves (eds.): Issues in Agent Communication. (LNAI 1916). Springer-Verlag (2000) 118-131
11. Grosz, B. J., Sidner, C. L.: Plans for Discourse. In: P. R. Cohen, J. Morgan, & M. E. Pollack (eds.): Intentions in Communication. MIT Press (1990) 417-444
12. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence 117 (2000) 277-296
13. Labrou, Y., Finin, T.: A semantics approach for KQML. In: Proc. of the Third International Conference on Information and Knowledge Management, ACM Press (1994) 447-455
14. Lochbaum, K.E., Grosz, B. J., Sidner, C. L.: Models of Plans to Support Communication. In: Proc. 8th Nat. Conf. on Artificial Intelligence. AAAI Press (1990) 485-490
15. Lochbaum, K.E.: The use of knowledge preconditions in language processing. In Proc. IJCAI-95 (1995) 1260-1266
16. Nodine. M., Fowler, J., Ksiezyk, T., Perry, B., Taylor, M., Unruh, A.: Active Information Gathering in InfoSleuth. Intl. Journal of Cooperative Information Systems 9 (2000) 3-28
17. Odell, J., Parnunak, H.V.D., Bauer, B.: Extending UML for Agents. In: Proc. Agent-Oriented Information Systems Workshop at the 17th Natl. Conference on Artificial Intelligence. AAAI Press (2000)
18. Pitt, J., Mamdani, A.: Communication protocols in multi-agent systems. In F. Dignum and M. Greaves (eds.): Issues in Agent Communication. (LNAI 1916). Springer-Verlag (2000) 160 - 177
19. Rich, C., Sidner C. L.: COLLAGEN: When agents collaborate with people. In M. H. Huhns & M. P. Singh (eds.): Readings in Agents. Morgan Kaufmann (1994) 814-819
20. Sadek, M. D.: A Study in the Logic of Intention. In Proc. 3rd Conf. on Principles of Knowledge Representation and Reasoning. Morgan Kaufmann (1992) 462-473
21. Searle, J.: What is a Speech Act. In: Black, M. (ed.): Philosophy in America. Cornell Univ Press (1965) 221-239
22. Shoham, Y.: Agent Oriented Programming. Artificial Intelligence (1993) 51-92
23. Wagner, T., Benyo, B., Lesser, V., Xuan, P.: Investigating Interactions between Agent Conversations and Agent Control Components. In F. Dignum and M. Greaves (eds.): Issues in Agent Communication. (LNAI 1916). Springer-Verlag (2000) 301-314-330.
24. Werner, E.: Cooperating Agents: A Unified Theory of Communication and Social Structure. In: L. Gasser and M. H. Huhns (eds.): Distributed Artificial Intelligence Vol II. Pitnam Publishing (1989) 3-36
25. Wooldridge, M., Jennings, N.J., Kinny, D.: The Gaia Methodology for Agent-oriented analysis and design. Autonomous Agents and Multi-agent Systems 3. Kluwer (2000) 285-312