Autonomous Agents and Multi-Agent Systems, 11, 273–305, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Normative Communication Models for Agent Error Messages

RENÉE ELIO AND ANITA PETRINJAK
Department of Computing Science, University of Alberta, 221 Athabasca Hall, Edmonton, Alberta, Canada, T6G 2E8

Abstract. An agent message is an attempted action upon the information state of the receiver that, if successful, would cause the receiver to move to a new information state. A model of normative communication can define when messages are not merely unsuccessful but instead are illegal or impossible actions upon the receiver's internal state. The model uses the preconditions of the other core message types, coupled with a model of task interdependencies, agent roles, and belief-desire-intention elements, to define the preconditions for sending a canonical *not-understood* error message. By defining the space of messages that are legal actions on an agent's internal state, a normative communication model also defines a set of 'reasons' that can accompany the error message. A *not-understood* error message signals a mismatch between agent interaction models and the accompanying reason opens the possibility for agents to realign their respective models. The paper discusses the matters arising from this possibility. This approach assumes that normative communication behavior reflects normative domain behavior. It also assumes that each agent accesses the normative model, in contrast with more centralized frameworks for defining normative interaction among agents and identifying interaction errors.

Keywords: agent communication, communication errors, interaction protocols, agent-based software engineering, cooperative problem-solving.

1. Introduction

Defining and standardizing a high level language for agent communication has been a cornerstone of the intelligent software agent paradigm. Indeed, Genesereth and Ketchpel [18] define software agents as "application programs which communicate with peers by exchanging messages in an expressive agent communication language." This definition is not as circular as it might first seem, for it implies the adoption of an agent theory that would enable such an ability. From a software engineering perspective, the adoption of a strongly typed agent communication language (ACL) is often driven by a need to have strong peer-to-peer communication among independent software entities, which in turn have some degree of autonomy in how they accomplish their respective tasks [34].

KQML (see [28]) and the Federation for Intelligent Physical Agents (FIPA)'s ACL [17] are two proposed ACL standards that have been widely adopted in agent research and development initiatives.[1] Agent systems can be developed with their own custom-designed ACL or with application-driven extensions of these two standards. An ACL specification defines a message syntax as well as set of core or primitive message types (e.g., *inform*, *request*, *query*), with reserved meanings. These primitive message types are often called the ACL's 'outer language.' Messages commonly use a keyword-value syntactic form, such as (*query :sender agent-i :receiver agent-j*

:conversation-id c12 :content α), where a field such as *:conversation-id* is used to associate independent messages with a conversation thread. The value α that follows the *:content* keyword is unconstrained by the ACL specification. Keywords such as *:ontology* and *:language* (e.g., Prolog, SQL) can designate the representational commitments that the sending agent followed when it constructed the *:content* value. Thus, in practice, the use of an ACL is a commitment to both an outer language of message primitives, and an inner language for expressing content.

For standards like KQML and FIPA's ACL, the semantics for message primitives are often defined as a set of pre- and post-conditions (loosely speaking) that reference the internal state of both the sending agent and the receiving agent. For example, the preconditions under which agent *i* can send agent *j* a *query* message concerning the truth value of proposition *p* might be defined as (a) agent *i* desires to know if *p* is true and (b) agent *i* believes that agent *j* knows the truth value of *p*. The post-conditions for *query* might be that agent *i* knows *p*, or alternatively, that agent *j* knows that agent *i* 'desires' to know *p*. Different ACLs embrace different semantics for seemingly very similar primitive message types (see [29] for a discussion of KQML's *tell* vs. FIPA's *inform*). The semantic commitments, of course, reflect the adopted agent theory, and this in turn influences the implemented agent architecture: terms like 'believe' and 'desire' must correspond to distinct internal states that determine behavior in some particular way.

Notions like 'believe' and 'desire' reflect two theoretical perspectives that have strongly influenced ACL semantics: (a) speech act theory and computational approaches to dialogue as planned actions (e.g., [1, 7, 39, 40]) and (b) Bratman's [3] theory of intention and rational action. A core idea is that an agent plans and selects an utterance (message) to have a particular, intended effect on the world, just as it would select any operator to achieve any goal in the traditional AI sense. In the case of communication, however, the 'world' that the agent aims to affect by sending a message is, at least in the first instance, the internal state of the receiving agent. And that world is not directly accessible. For example, an agent cannot directly observe if its *inform* message had its intended effects on the internal state of the receiving agent, no matter how those effects are defined. And as the semantics for the *query* example above illustrate, even the pre-conditions for sending a message are couched in terms of beliefs about the inaccessible world that defines the receiver's internal state. Because of the inherent uncertainty in communication from this perspective, philosophical analyses and formal semantics for communication employ notions such as success conditions [40], feasibility conditions and intended rational effects [17], and attempts [6]. One simple example of a success condition (although quite important in implemented systems) is that the receiving agent actually *receives* the sent message; another is that that receiving agent can interpret the *:content* of the message.

Our point of departure here is this core premise about agent communication: that a message is an attempted action upon the information state of the receiver that, if successful, would cause the receiver to move to a new information state. This is consistent with the perspective of update semantics [44] and dynamic semantics [20] that the meaning of a sentence is the change in the receiver's information state that the utterance of that sentence brings about. From this, we explore a fairly simple idea. Namely, it should be possible for a given agent system to define a *model of normative communication*, such that deviations from this model are not *merely* unsuccessful actions in the world, but are rather, in some basic sense, *illegal* or *impossible* actions upon this world. A normative communication model would specify the set of legal computational actions upon elements of the receiving agent's internal state, realized as messages from a sender to that receiver.[2] Any communication outside that space would be an illegal or impossible action that the sender is attempting on an element of the receiver's internal state. A familiar example is an attempt to divide a number by zero: division is a legal operation, but not when its denominator is zero. An attempt to do the latter will generate a run-time error. For a particular agent application, we could imagine analogous cases, namely that it is legal – in principle – for agent i to inform agent j about proposition p, but *not* about proposition q. Or that it is legal for agent i to *inform* agent k about p, but not agent j. Upon receiving an inform message about proposition p, agent j might still not adopt belief in proposition p for various reasons. But this is different from the case where it was illegal – by some model – for i to attempt an update on j's belief state with an *inform* message, for a particular proposition p or perhaps for any proposition.

The issues here are two-fold. First, what are the components of normative communication model that would define a space of useful communication behavior and errors for multi-agent systems? Second, what is a pragmatic and structured way to represent and use such a model in a multi-agent system? A normative communication model enforces good software engineering principles for multi-agent systems, particularly those that depend heavily on communication for planning or decision-making. Both KQML and FIPA's ACL anticipated the need for an error message type that an agent could send in response to a message that it received but could not (for some reason) successfully process. We examine those message types in more detail shortly. However, there has been little consideration of possible models for generating communication error messages and for guiding agent behavior, when a communication error message is received.

The work we describe here presents some core elements for a normative communication model that defines the preconditions for sending an error message, using task interdependencies, agent roles, and belief-desire-intention elements in a unified fashion. In doing so, we identify a number of issues that emerge in committing to such a model and using it to govern agent behavior. We first begin with a closer look at the error primitives in both KQML and FIPA's ACLs. This will clarify the character of a normative communication model and why it is pragmatically and theoretically important for agent systems. We then present the elements of the normative communication model we have tested in a multi-agent system. Next, we describe the model's declarative and procedural representations, which delimit and validate agent communication actions. We then consider how such a model defines cases for generating error messages and in particular, how the same model can constrain how an agent *responds* to a received error message. This brings us back to the view that communication acts are attempted accesses, updates, or revisions on elements of another agent's state. We conclude by considering how other frameworks have approached communication error handling and the relation of this work with some software engineering issues.

2. Message primitives for error conditions

To clarify and motivate the role we see for a normative communication model, we first consider the error message primitives in KQML and FIPA's ACL. KQML has two such primitives. The first is called *error* and has the syntactic form (*error :sender i :receiver j :in-reply-to cid1 :reply-with cid2*), where *i* and *j* are agents, and *cid1* and *cid2* are unique identifiers to designate particular messages. Here, agent *i* might be sending this error message as a response to agent *j*'s message *cid1*, which (for example) might have been a request to insert some new content into a database. According to KQML specifications, the occasion for an agent to send an *error* message is described as follows:

> This [*error*] performative suggests that the *:sender* received a message that it does not comprehend. The cause for an error might be: (1) a syntactically ill-formed message, (2) the message has wrong performative parameter values, or (3) it does not comply with the conversation protocols [27, p. 22].

Protocols are predefined multi-message sequences (e.g., an auction protocol, a bidding protocol) which specify which message primitives are allowed to follow each other [46]. In this sense, a protocol constitutes a normative communication model, albeit a shallow one from our perspective.

KQML also defines a *sorry* message, with the same syntactic form as *error*, but with a subtle and important difference:

> This [*sorry*] performative indicates that the *:sender* comprehends the message [designated with *cid1*], which is correct in every syntactic and semantic sense, but has nothing to provide as a response…when an agent uses *sorry* as a response to [some received message] this means that the agent did not process till the end the message to which it is responding to, e.g., an agent that responds with a *sorry* to *insert* never inserted the *:content* into its KB [27, p. 24].

The *sorry* message, unlike the *error* message, implies a kind of operator legality for the original message (e.g., the original request to do some database operation) but indicates failure insofar as the intended world change did not, for some reason, occur.

In FIPA's ACL, there is a single error message primitive called *not-understood*. The form of this message is (*not-understood :sender j :receiver i :content c*), where *i* and *j* are agents. Like the *sorry* and *error* messages, 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 two parts. The first part is the entire message that triggered the *not-understood* reply (e.g., an entire, just-received *inform* message from *i* to *j*) and the second part is 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. [The

> 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 [17].

This 'reason' could, in principle, cover both situations that define KQML's *error* and *sorry* messages. The specification of a reason invites an approach to handling a communication error based upon the interpretation of this reason, either by the agent system designer or (more challenging) by the agents themselves.

Message types like *error*, *sorry* and *not-understood* are quite important for agent interaction. As we noted above, an agent who sends a message is aiming to change an aspect of a non-deterministic world that is not directly accessible, namely the internal state of the receiving agent. Hence, a message like *not-understood* is quite a pointed response back from that inaccessible world, signaling that the action was not merely unsuccessful but – by some model – illegal or undefined in the first place. Even if a particular application called for its own custom-designed ACL, rather than the adoption of one of these standards, it seems likely that it would include some message primitive to serve the same role of *error* or *not-understood*. This would just be good software engineering. With this in mind, we use FIPA's *not-understood* message as a canonical error message because its inclusion of a reason opens up a number of theoretically interesting avenues, which we explore in more detail below.

There are other reasons why we think it is theoretically and pragmatically useful to take a serious look at explicitly defining the preconditions for sending a *not-understood* message. First, thinking about when to send a *not-understood* message is a different way of thinking about what it means to *understand*, i.e., for an agent to proceed with assimilating a message. We can think of understanding as the success of a message's 'extended perlocutionary effects,' i.e., how a message alters an agent's state, *such that the agent behaves differently* for having received and assimilated the message [13, 36]. This is consistent with the view of a sentence's meaning as a function on information states [20, 44]: if an agent is in a new information state, then that agent has the potential to behave differently. Second, it is generally acknowledged that an ACL specification does not, by its very nature, address the matter of what agents actually exchange messages *about*, i.e., what fills the *:content* field of a message [14, 21]. Neither do shared ontologies and shared content languages. For a wide range of applications, particularly those that are cooperative or coordinated in nature, it seems that what agents must exchange messages *about* is their progress on interdependent tasks. The abstract specification of task interdependencies and task progress can serve as part of a jointly held conversation 'policy' about goal achievement [13, 19]. Such a policy can serve the same role as a protocol, by defining whether it is normative to send a particular message type at a particular point in time and by defining the normative content of a message in different contexts. That is the approach we incorporate into the normative communication model framework that we describe here, which requires a declarative approach to modeling tasks and agent interdependencies.

Finally, if we seriously regard the notion of agent as a kind of programming abstraction [41], then *not-understood* can be viewed as signaling a 'run-time error' for a particular class of computations (message generation) upon some object (an element in the internal state of the receiver). The 'constructive reason' that is part of the *not-understood* message can be viewed as an error code (e.g., 'division by zero'). Shoham [41] proposed that agents be programmed in terms of their mentalistic, internal states: their commitments, their desires, and their beliefs. Since communication is an action attempt on those elements, we are proposing to define communication error conditions in the same fashion: as illegal action attempts upon the beliefs that (partially) define an agent.

In addition to the software engineering mileage that a declaratively specified normative communication model offers, some interesting theoretical issues also arise. If a *not-understood* message is regarded as a run-time error, then at the very least, the agent system should halt. Ideally, the system designer ought to be able to use the 'constructive reason' to debug the system. But it is natural to ask if there are normative reactions and responses an *agent* can make, when it receives a *not-understood* message. This is closer to the spirit implied by the FIPA specification that appeals to a 'cooperative agent' that attempts to 'explain misunderstanding constructively.' From our perspective, if agent *j* asserts that message *m* is *not-understood*, it does so by appeal to some model. The sending agent *i*, by virtue of having generated the message *m*, must be following some other model – otherwise it would not have sent message *m* in the first place. What are the minimal effects of receiving a *not-understood* on agent *i*? At the very least, agent *i* ought not to send message *m* to agent *j* all over again. Agent *j*'s 'constructive reason' opens the possibility for agent *i* to modify its model, so that this does not happen. Of course, it may instead be the case that agent *j*'s model is 'wrong' – agent *j* ought to regard message *m* as legal – and agent *j* should revise its communication model so that it accepts message *m*. The issue we are hinting at here is, crudely put, whose model is taken to be the correct one. We have no general solution for this, but we discuss some preliminary ways that the communication model itself can define policies for this decision.

Before embarking on the design of normative communication model, there must be a commitment to some particular agent theory, for it is the assumptions of that theory that dictate key aspects of the model. For this work, an agent is defined by an internal information state (belief state) and by its functional capabilities within some application domain. We assume that a portion of the internal belief state is propositional in nature, stated in an abstract, domain independent vocabulary. This vocabulary will serve as the inner content language. An agent's functional capabilities are (a) the ability to solve the task to which it has been assigned, or to determine it is not solvable, and (b) the ability to generate and assimilate messages specified in a high-level ACL. An agent is autonomous, in the sense it has whatever domain-level capabilities it needs to transform its task from an initial state to a final state (succeeded or failed) and to determine if its task is relevant or impossible (again, concepts defined by the abstract inner language). We also assume agents can be proactive, initiating their own problem solving work rather than waiting to be invoked by a centralized authority. Both these assumptions create a need for peer-to-peer communication, as an agent attempts to acquire or provide information so that the overall task can proceed. We also use simple elements of belief-desire-intention theory (see [49]) to relate agent state to agent behavior. With this view of agency in mind, we next examine the kinds of normative communication errors our model aims to define.

3. Occasions for not-understanding

To illustrate the general intuitions behind our approach, the top portion of Table 1 presents our choice for an *inform* schema, which is based on FIPA's feasibility preconditions and intended rational effects for *inform*. We also include what might be called contextual relevance conditions, e.g., that the receiver of the *inform* wishes to know α [4, 26, 39]. Finally, we specify certain success conditions [40] that stipulate (some) conditions that must hold for the *inform* action to be successful. The resulting schema is a blend of the pragmatic conditions from the semantics proposed for KQML and the feasibility conditions of FIPA that describe the minimal requirements on the state of the sender to send such a message. In this schema, the intended effect of an *inform* action is that the receiver adopts belief in α i.e., that the receiver's internal state includes an element that corresponds to belief in α.[3]

Table 1 shows six preconditions for this particular *inform* schema. Although these are classified as feasibility, relevance, or success conditions, we regard them as jointly defining a normative *inform* action. An *inform* is legal to send when all these conditions are holding. More precisely, it is normative for agent i to send agent j an *inform* message when agent i's internal state includes all the beliefs listed in the *inform* schema.

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

< inform i, j, α >	
Success conditions:	i. α is interpretable by j
	ii. j's state concerning α can be updated
	iii. j's state concerning α can be updated by i
Feasibility preconditions:	iv. i knows α
	v. i believes j has no position on α
Contextual relevance conditions:	vi. i believes j desires to know α
Intended effect	j adopts belief in α

< inform i, j, α > is not-understandable to j wrt a normative communication model if

(i) α's predicate is not in a commonly shared ontology
The intuition: "I don't know what α means."

(ii) j's position on α cannot be updated/revised
The intuition: "My belief about α cannot be changed."

(iii) j's position on α cannot be updated/revised by a communication act from i
The intuition: "*You* cannot change my belief about α"

(iv) j cannot explain why i would know α
The intuition: "How is it that *you* know α?"

(v) j cannot explain why i believes j has no position on α.
The intuition: "Why do you believe I do not *already* know α?"

(vi) j does not desire to know α
The intuition: "*Why* are you telling me α? It would have no impact on my behavior."

Conversely, an *inform* message is *not-understood* if, from the receiver's perspective, one or more of these conditions is not holding. By defining these as the conditions for a normative *inform* action on the receiver's internal state, we can reverse them and define deviations from them as occasions for sending a *not-understood.*

The lower portion of Table 1 presents six occasions for sending a *not-understood.* These occasions are derived from the six preconditions in the *inform* schema. The first three are concerned with the propositional content of the message itself, the particular illocutionary force (message primitive) applied to the content, and the role of the sender to the receiver. In structuring these cases, we use the idea of predicate classes, agent interdependencies, and agent roles.

Case (i) concerns whether it is normative to make the propositional content α the object of *any* communication action, i.e., to use it in the *:content* field of any message. To cover this case, our model defines predicates as belonging to one of two classes: 'public' or 'private.' Only propositions involving public predicates can appear as *:content*. In our current model we use this distinction in a simple sense, to cover the notion of a shared ontology. In traditional programming, this loosely corresponds to private and public methods, or global variables vs. local variables. More generally, the private–public distinction might be used to implement privacy constraints about the exchange of particular information among particular agents.

Case (ii) can be viewed as loosely mapping to the distinction between constants and variables in traditional programming. Our adopted agent theory defines an agent, in part, by an internal state populated by propositional beliefs. We assume that some of those elements can change their truth status during run time, while others may not. Our communication model puts the former type in a class it calls 'dynamic' and the latter type in a class it calls 'static.' For example, the propositional belief element *(agent-for agent-j task-12)* represents the belief that agent *j* is the assigned agent for task-12. If *agent-for* is a member of the dynamic predicate class, then, in principle, the assignment of agents to particular tasks can change during run time and one agent might inform another agent about such a re-assignment. This would be a legal belief update or belief revision operation upon the receiving agent's state. If agents do not change their tasks during run time, then *agent-for* must be in the class of static predicates, and such an *inform* would be an illegal operation.

Our model further classifies dynamic predicates into two subtypes: 'internal' and 'external'. The intuition here is captured in Case (iii), which corresponds to whether agent *i* can 'change its mind' about the truth status of α only through its own internal reasoning, or whether that truth status can be changed as a direct result of an external communication action taken by another agent. For example, our model classifies predicates such as *desire* and *intend* as internal dynamic predicates and in doing so, prohibits them from appearing in the content of certain message primitives. By such a model, agent *i* may not *inform* agent *j* what agent *j* believes, intends or desires. This embodies 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 α." [1]). But this distinction applies more generally to other propositions in *j*'s belief state, namely any proposition for which only *j* can determine a truth status. This distinction maps loosely to the programming distinction between a procedure's locally defined variables and the values passed to it as parameters.

Cases (iv)–(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 either a global or a partial model of interdependencies among tasks and the agents assigned to those tasks [9, 11, 22, 32]. Using such information, 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 concerning a message's extended perlocutionary effects. Presumably, i intends that j adopt belief in α so that j's behavior will change. If believing α 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 will be quite important for agent i to learn that the *inform* message that 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 process the *inform* message in terms of the models it has about tasks and agents. This could lead to a domain-level error situation, brought about because an *inform* action did not have its intended effect. This reflects the spirit of KQML's *sorry* primitive – a syntactically correct message with no consequence. FIPA's ACL has a similar primitive called *refuse*, which also signal's that the intended effect – typically a request for some action – did not occur. We see the *not-understood* message as serving a qualitatively different role than *refuse*, namely to indicate that the attempted communication or domain action violates some underlying model.

We wish to make a few additional points at this juncture. First, we have not explicitly included the case of protocol violation, which is a 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 can be more easily recognized as syntactic violations at a shallow level of processing (i.e., a particular message sequence is not permitted), without complicated models of agent relationships or message content. In our framework, a normative communication model that is derived from task and agent interdependencies can yield protocol-like behavior, to the extent that it defines what message is legal or plausible for an agent to send to another agent at any point in time.

Second, we use the term ‘explain’ in Table 1 to signify that a received message must be consistent with the receiver's model of normative communication, based on a model of task and agent interdependencies. 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. However, we do return to the matter of just how much reasoning an agent must or might do, in formulating *not-understood* reasons and in responding to *not-understood* messages that it receives.

Third, Table 1 is not intended as a statement of what is complete or necessary as semantics for an *inform* message. We use the *inform* preconditions in Table 1 to define the corresponding preconditions for a *not-understood* response. An ACL designer might have different preconditions for this message type, in which case the occasions for *not-understood* would be different. The idea is that an ACL designer can exploit the preconditions of the ACL's core messages as a starting point for defining *not-understood* occasions.

Finally, it is natural at this point 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 *ad infinitum*. This is the matter we noted earlier, concerning which agent model is going to be taken as correct. In a later section, we show how the underlying normative communication model can be used to avoid this.

In our work to date, we focus on message types that take propositions as message content, namely *inform* and *disconfirm*.[4] We also handle a form of *query* (namely, *query-ref*), whose semantics under the FIPA specification correspond to a *request* for an *inform* action. FIPA includes *refuse* as a possible response to a request for an action, although a *query* generates a *not-understood* by our analysis in Table 1. For brevity's sake, we limit our discussions and examples in the remainder of this paper to *inform*, expanding to cases of *query* and *disconfirm* when appropriate.

4. Elements of a normative communication model

There are three main components to a normative communication model that support the generation of *not-understood* messages for the cases listed in Table 1. The first is a language to describe tasks and interdependencies among tasks, from which interdependencies among agents assigned to those tasks can be derived. TÆMS [11] is an example of such a task modeling language and we adopt several of its distinctions. The second component is the definition of predicate classes that constrain what inner message content can legally appear with particular outer message primitives (*inform, query, disconfirm)* in various contexts. The third 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 their tasks. It is these propositions that agents will access, update, or revise through communication actions during their interactions. Particular exchanges, revisions, and updates are then defined as legal, *given* this normative communication model. The remainder of this section aims to present just enough detail about the framework model to ground our intuitive examples from Section 3 and to support our later analysis of responding to *not-understood* messages during run time.

4.1. Task models

As a motivating example, Figure 1 presents the conceptual decomposition of a simple airline reservation application. Our main concern is the communication patterns that arise when subtasks in such a decomposition are distributed among independent agents. The accomplishment of any particular subtask will likely require various resources. Here, we consider only informational resources, which is sufficient for the communication model we construct. For example, an agent responsible for displaying the information about the airplane tickets (‘Obtain Customer Information’) needs information as a sort of resource, otherwise it is not able to successfully perform its task of displaying it.

The task structure in Figure 1(a) is modeled at a more abstract level, illustrated in Figure 1(b). Following [11, 42, 43], the solution method for a task is specified either as 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. Every task and method definition includes pre- and post-conditions, stated as Boolean constraints on domain-specific variables. These specify when a task can be initiated and deemed successful, respectively. For example, in the airline ticket application, one of the preconditions for initiating the task of obtaining user information is that at least one

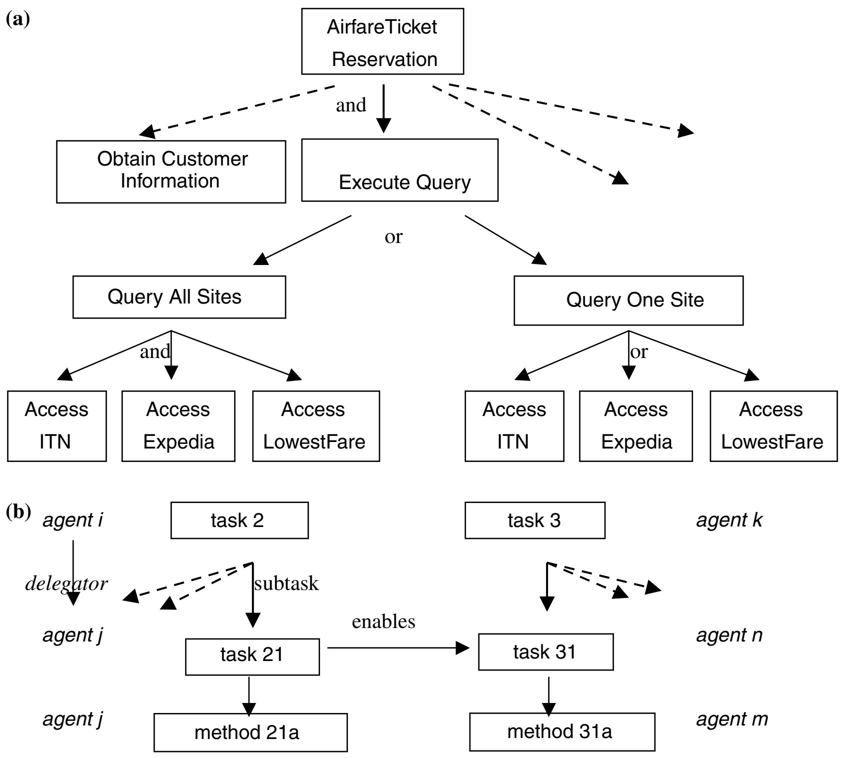

Figure 1. (a) Partial decomposition of a domain task and (b) abstract task structure elements.

of the departure and return dates is unknown; a post-condition is that both 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 a constraint is satisfied. In our domain-independent language, the preconditions are regarded as resources that an agent requires to begin work on its task.

The task structure is a declarative element of the normative communication model and we assume that all agents have access to the entire model. This is not strictly required, e.g., agents might have access to just those portions of the task structure that involve agents with whom they must or may communicate. Agents use the task model (coupled with certain axioms, described in Section 4.3) to derive what other agents must know or need to know. The completeness of this model therefore plays an important role in accurately defining the range of legal or plausible messages exchanges.

4.2. Predicate classes

As we noted earlier, our normative communication model assumes that an agent has some internal state, which includes a belief state with propositional elements. Messages are attempted operations on this state, and we consider a limited set of such operations. An *inform* message can be viewed as an attempted belief update, a *disconfirm* as an attempted belief revision, and a *query* as an attempted access operation. Each message type takes a proposition α as its content. Earlier we introduced a classification scheme for the predicates that appear in these propositions. This scheme, coupled with other elements of the communication model, serves as a first step in defining the legal communication actions that can be performed on propositional beliefs.

Table 2 lists a set of predicates that are part of our domain-independent inner language. Having outlined the intuition behind these predicate classes earlier, we recap only a few key distinctions here. The private–public property distinguishes the propositions that *may* appear within a message's content from those propositions that *may not* appear as message content. The static–dynamic property specifies which propositions can have their truth value changed during run time. The internal–external property – as a further classification of dynamic predicates – designates how a proposition's truth value can be revised or updated, either through an agent's own computations (internal) or as a consequence of a communication received from another agent (external). Predicates are classified according to each of these properties. For example, the predicate *relevant* is classed as public–dynamic–external (Table 2a): a proposition that some task is relevant can appear as message content, can be updated or revised during the course of agent interaction, and such a revision or update may be the consequence of a communication action. Private–dynamic–internal predicates (Table 2b) appear in propositions that correspond to whatever internal information an agent maintains for its task computations. The intend, desire, and belief modality predicates constitute a subclass under public–dynamic–internal (Table 2c), indicating that they can appear in message content propositions, (public), belief state propositions with these predicates change during run-time

Table 2. Classification of core predicates for a sample communication model.

- (a) public/dynamic/external
  - relevant(t): task *t* is to be performed
  - resource(r, v): resource *r* has the assigned value *v*
- (b) private/dynamic/internal
  - class = processing
    - done(t): a succeeded task *t* is in its final state and its post processing is finished
    - valid-result(a, t, r), have-resource(a, t, r), have-all-resources(a, t): *a* is an agent who has the beliefs that *r* is a valid result for task *t*, *r* is a resource for task *t*, and all resources are available for task *t*, respectively
- (c) public/dynamic/internal
  - status(t, s): the status of task *t* is *s*, *s* = {not-attempted | possible | not-possible | failed | succeeded}
  - result(r, v): result *r* has the assigned value *v*
  - class = modality (*see text*)
    - intend(a, α), desire(a, α), believe (a, α)
- (d) public/static/—
  - class = predicate class (*see text*)
    - private(p), public(p), static(p), dynamic(p), internal(p), external(p)
  - class = task-structure
    - has-subtask(t, s), subtask-of(t, s), agent-for(a, t), enables(t, s), enabled-by(t, s)

(dynamic), but not through an *inform* or *disconfirm* message (internal). The implementation of the dynamic–internal vs. dynamic–external distinction cannot be a table look-up. Agent *i* may inform agent *j* about what agent *i*'s intends (for this is knowledge agent *i* must have). But agent *i* cannot inform agent *j* about what agent *j* intends. We have not done a formal logical axiomitization of these kinds of constraints, but it could be done in the proper modal logic.[5] There are many possible implementations of such constraints; our own approach takes the form of a context-sensitive grammar that validates incoming messages, which we present in a later section. Finally, Table 2d indicates that propositions corresponding to beliefs derived from the task model (e.g., *enables*, *agent-for*) can serve as message content (public), but that the truth value of such propositions cannot change during run-time (static). Again, a different model might well indicate that such propositions are open to revision during agent interaction.

4.3. *Semantics and inference rules*

The semantics of these predicates must be reflected in some coded interpretation or set of explicit reasoning rules. Such rules unite task properties with agent properties. Some examples of these rules from our current model (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*." "*If the agent a is responsible for task t, then it knows the status of t*." "*If task x is irrelevant or failed or not-possible, and task y is a subtask of that task, then task y is irrelevant*." 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. The derived beliefs from these axioms create propositions that serve as preconditions for sending messages. Hence, they are central to the specification of a normative communication model.

Further discussion of our axioms and inference rules is outside the scope of this paper and not central to the main points we examine here. 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. Any implementation must ensure that, during execution, agent belief states and behavior are consistent with the axioms that define the agent semantics.

4.4. *Unifying problem solving and communication*

An agent's computations on its domain-specific information states can be carried out in any fashion, as long as they ultimately produce propositions that use the abstraction vocabulary (e.g., propositions based on predicates appearing in Table 2 a, c and d). Such propositions are the possible objects of communication acts. In our particular system, agents do this by using operators such as the following (in English gloss):

```
If      I intend task t
        & 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.
```

Such operators need not be the same for all agents. We also assume that each agent has a communication plan module that handles the generation and assimilation of the message types. Communication plans are triggered in two ways in our system. 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 obtain information about resources that it requires for taking its task from a *not-attempted* to a *possible* state(see [32]). 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. This latter assumption is important for applications in which agents are not invoked under some strict calling hierarchy, but have some autonomy in initiating their tasks and optimizing their interactions.

The communication plans in our system implement message schemas, such as the one shown in Table 1. An agent's communication module also creates the necessary data structures for maintaining conversations with other agents. Incoming messages are recognized either as completing or continuing an on-going conversation, or as initiating a new conversation, provided that such messages pass the *not-understood* filters (described below). Our focus is not on the details of this communication layer *per se*, but rather on how the communication model allows it to assimilate or generate *not-understood* messages.

5. Representing a normative communication model

Agents must be able to access and interpret a declarative representation of the normative communication model. We use XML for this representation and the model divides into two conceptual parts. The first is the task structure. Table 3 shows a portion of the XML representation for the task “Execute Query” from Figure 1. The notation indicates that this is a subtask (“component”) of a single parent (task T011, the unique task identifier for the root node task “Airline Ticket Reservation” in Figure 1a). Its assigned agent is “ag3.” This task is accomplished through the successful completion of either of its two sub-tasks (an “or” decomposition). These are tasks T035 and T036, which correspond to “query all sites” and “query one site” in Figure 1. The remaining notation expresses pre- and postconditions that define whether the task can, in principle, be initiated and when the task can be declared done. Here, one of the postconditions for successful completion indicates that a data structure for holding ticket information has at least one entry (is not null).

The second part of the XML representation is what we call the explicit communication ontology. This is a representation of the remaining elements that define the normative communication model: (a) all predicates that can appear in an agent’s belief state, classified along the public–private, dynamic–static, and internal–external dimensions; (b) all agent roles derivable from axioms relating task properties to

Table 3. Excerpt from the XML representation for “Execute Query” task.

```
<Task>
        <property id = “T022”/>
        <property name = “Execute Query”/>
        <property component_of = “T011”/>
        <property parent_list = “T011”/>
        <property children_list = “T035, T036”/>
        <property agent = “ag3”/>
        <Decomposition>
                <property child_type = “task”/>
                <property cardinality = “2”/>
                <property choice = “or”/>
        </Decomposition>
        <Precondition>
                <property id = “origin”/>
                <property expression = “origin! = null”/>
        </Precondition>
        <Precondition>
                <property id = “destination”/>
                <property expression = “destination! = null”/>
        <!— remaining preconditions removed for brevity—>
        <Postcondition>
                <property id = “ticket_table”/>
                <property expression = “ticket_table! = null”/>
        </Postcondition>
        <!— remaining postconditions removed for brevity—>
</Task>
```

agent properties, (c) modalities (agent attitudes towards propositions), and (d) the allowable outer message types (here restricted to *inform*, *disconfirm*, *query-ref*, [6] and *not-understood*). Our reason for classifying our modality predicates (intend, believe, desire) separately will become clear shortly.

With these elements in place, it is possible to define a grammar that corresponds to the declarative representation of this model. Table 4 shows a context sensitive grammar for an *inform* message (for convenience, we have defined two versions of inform messages, *inform-1* and *inform-2*, in this grammar).

An example of a schematic *inform-1* that this grammar accepts as valid is: (*inform sender receiver delegator* (*task-of*$_{\text{sender}}$ *relevant*)). This message is consistent with the model specifications that it is normative for an agent *i* (as sender) to inform agent *j* (as receiver) of agent *i*'s task relevance, if agent *j* is performing a subtask for agent *i* (i.e., *i* has the relationship of delegator to *j*). Using Figure 1b task structure as an example, the grammar allows *i* to inform *j* that task-2 is not relevant. Such a message is legal, because agent *i* can send messages with propositional content about task relevance (the *relevant* predicate is classed as public) and propositions about task relevance can be updated via communication acts (*relevant* is also dynamic-external). This *inform* message is also 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 or accept the following *inform-1:* (*inform sender receiver delegatee* (*task-of*$_{\text{receiver}}$ *relevant*)). Given the Figure 1b task structure, agent *j* cannot inform agent *i* about the relevance of agent *i*'s task (task-2): agent *j*'s role to agent *i* is delegatee, not delegator. More generally, this grammar specifies it is invalid for an agent to inform another agent about its own task relevance, if those agents do not have some kind of interdependency (although a different model, and corresponding grammar, might permit this).

An example of a schematic *inform-2* that this grammar generates is: (*inform sender receiver delegatee* (*sender believes*)). This second type of *inform* message captures the notion that a sender can inform a receiver about the sender's attitudes towards propositions, but not about the receiver's attitudes about propositions. Again, the

Table 4. A grammar for legal inform messages (*s-to-r* = *sender-to-receiver*).

Terminal symbols:	inform, sender, receiver, task-of$_{\text{sender}}$, task-of$_{\text{receiver}}$, delegator, delegatee , enabler , enablee, intend, believe , desire, relevant, resource, result, status
Non-terminal symbols:	inform-message, inform-1, inform-2, role-of$_{\text{s-to-r}}$, content, external, modality
Rules:	
inform-message	⇒ inform-1 | inform-2
inform-1	⇒ (inform sender receiver role-of$_{\text{s-to-r}}$ content)
role-of$_{\text{s-to-r}}$	⇒ delegator | delegatee | enabler | enablee
content	⇒ task-of$_{\text{sender}}$ | task-of$_{\text{receiver}}$
role-of$_{\text{s-to-r}}$ task-of$_{\text{sender}}$	⇒ role-of$_{\text{s-to-r}}$task-of$_{\text{sender}}$ status
delegatee task-of$_{\text{sender}}$	⇒ delegatee task-of$_{\text{sender}}$ result
delegator content	⇒ delegator content external
external	⇒ relevant | resource
inform-2	⇒ (inform sender receiver role$_{\text{s-to-r}}$ (sender modality))
modality	⇒ intend | believe | desire

communication model that corresponds to this notion requires that *some* direct role exist between the sender and receiver (e.g., the grammar would not generate this message from agent *i* to agent *m* in Figure 1). A different normative communication model might allow any agent to tell any other agent what it intends or desires, whether they have an interdependency or not.

The grammar in Table 4 essentially defines four valid types of informs and it is straightforward to represent a context-free version of this grammar in XML. When a message arrives, agents parse it to extract the core elements defined by the grammar, and use the XML rules to determine if the parsed message fits one of the prescribed formats. The grammar embodies constraints about public, dynamic, 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 define grammars like the one in Table 4 for *disconfirm* (similar to inform) and *query-ref*.

The declarative representation of the grammar coupled with a general interpretative routine allows an agent (a) to generate all legal and pragmatic messages between itself and another agent, and (b) to validate whether the general form of an incoming message passes various types of understandability filters. The validation process serves as the general run time check on the communication operations that agents attempt on each other's belief states. The grammars also serve an important function as well – they constrain the kind of responses that an agent can make to a *not-understood* message.

6. *Not-understood* messages and mismatched agent models

Given these declarative and procedural representations for a normative communication model, a message validation and assimilation routine can be designed to parse an incoming message and determine whether the preconditions for a *not-understood* response have been met. Essentially, the routine checks these four cases, which cover the six intuitive cases outlined in Table 1:

1. Is the predicate in the *:content* proposition a public predicate?
2. Is the illocutionary force (message primitive) applied to the *:content* proposition allowed?
3. Does the sender hold the proper role relative to the hearer, in order to apply this illocutionary force to the propositional content?
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 form of a *not-understood* is (*not-understood :sender j :receiver i :content* (*m reason*)), where the *m* is the original message. The *not-understood* reason that we generate takes one of the following forms:

```
(not-understood: sender j :receiver i
   :content  (m (not (public ⟨predicate p of m's content⟩))) |
             (m (not (dynamic ⟨predicate p of m's content⟩))) |
             (m (not (external-dynamic⟨predicate p of m's content⟩))) |
             (m (not (permissible-role sender receiver))) |
             (m (not α)))
```

The first three reasons correspond to violations of what agent *j*'s normative communication model defines as legal combinations of outer message primitives with inner message content (i.e., what agent *i* attempted on a particular propositional element within *j*'s state). The fourth reason asserts that agent *i* did not have the correct interaction role relative to *j* in order to send message *m* with its given propositional content. The last reason is a proposition α that corresponds to one of the preconditions that the receiver believes ought not to be holding on the part of the sender. For example, this case might correspond to "It is false that I desire the result of task-23." This proposition must be stated in the abstraction language for representing beliefs, be part of the finite set of propositional beliefs that any agent can have, and directly or indirectly correspond to one of the preconditions for the original message.

At this point, we could consider the matter of generating and structuring *not-understood* messages as done. With this sort of normative communication model, agents have well-defined preconditions, grounded in an interaction model, for generating *not-understood* messages. If we regard *not-understood* messages as true run-time errors, then such a message would bring an agent system to a halt, or at least stop further interaction between two particular agents. In principle, if all agents are following the same normative communication model, then *not-understood* messages should not occur. If and when it happens, a *not-understood* message would reflect a bug in agent design or implementation, or a change in some agent's normative communication model, that renders it inconsistent with another agent's model. This might happen if agents could be reassigned to tasks during run-time (thus supporting different beliefs about what are legal and plausible messages to send among each other).

What happens when agent *i* receives a *not-understood*? Minimally, we want its internal state to change in some way, so that it does not resend message *m* over again to agent *j*, forever causing agent *j* to send back *not-understood*. Recall that the reason included with a *not-understood* is intended to be a 'constructive explanation' about why the original message *m* was not understood. A *not-understood* message from agent *j* to agent *i* can be viewed as a composite of two primitive messages: an *inform* that the intended effect of message *m* has not occurred and an *inform* (or *disconfirm*) from agent *j* to agent *i* of some additional proposition α, where α is the one of the preconditions of a normative inform schema that *j* believes to be false. This is the 'constructive explanation.' But there are a few matters to consider here. Just how much diagnosis must agent *j* perform to determine the core error in agent *i*'s model, so that the right or most constructive reason is sent along with the *not-understood*? If there is more than a single possible reason, should they all be sent? And what must or might agent *i* do with the reason it receives? It could adopt it and thereby update or revise its global interaction model, revise its particular model for interacting with *j*, or instead send another message to *j* in an attempt change agent *j*'s model and thereby make the original message *m* legal for *j*. But this immediately raises the question of whose model will carry the day – the model of agent *i*, who generated message *m*, or the model of agent *j*, who received it?

We do not have a general answers for these, but there are simple ways in which the underlying communication model can constrain what agent *i* is allowed to do, having received a *not-understood* message. The more complicated issues require a position on just how much reasoning the individual agents can – or ought to – perform, in order to align their mismatched models. The cases below illustrate how our current framework controls and resolves *not-understood* conversations and how some of these more thorny issues arise.

6.1. Allowable message content

One of the basic *not-understood* cases is when an agent sends message content that cannot be resolved within the shared ontology. This means that the intended operation on the receiver's internal state cannot occur and the receiver's behavior cannot be affected by the message. Consider this possible message conversation (for presentation purposes, we do not repeat the entire message in the *not-understood* content tuple, but designate it by a label):

Case A

message A1	(inform *i j* (have-resource (*i*...)))
message A2	(not-understood *j i* (⟨message A1⟩ (not (public have-resource))))
X message A3	(not-understood *i j* (⟨message A2⟩ (public have-resource))))
X message A3′	(disconfirm *i j* (not (public have-resource)))

In message A2, *j*'s reason is that *j* believes that propositions involving the predicate *have-resource* cannot be exchanged. Now, there are two possible responses that *i* could make. The first is message A3, in which *i* informs *j* that *j*'s *not-understood* message is not understandable to *i*. However, the model prevents *i* from generating A3, because A3 is equivalent to *i* informing *j* that *i* believes α, where α is (*public have-resource*). However, *i* already knows (from message A2) that *j* believes $\sim\alpha$. So a feasibility condition for the implicit *inform* within A3 is not satisfied, and this disallows A3.

Can *i* instead disconfirm *j*'s belief about whether or not *have-resource* is in the class of public predicates by sending message A3′? That depends on whether the predicate *public* itself can appear in propositions used as direct *inform* or *disconfirm* message content, and by Table 2d, it cannot (it is not classed as dynamic-external). So agent *i* is prevented from generating A3′. Instead, it must instead adopt the belief *have-resource* is not a public predicate – at least for interactions with agent *j* – and this prevents it from resending A1 again.

Suppose that, according to *i*'s model, the predicate *public* is in fact classed as a dynamic–external predicate. Then it might send message A3′, and thereby attempt to change how agent *j* classifies the predicate *have-resource*. If it did, agent *j* would send a *not-understood* back to A3′, indicating that by its own model, the predicate *public* is neither dynamic nor external. Agent *i* could not 'argue' further on this point, since there are no other legal informs or disconfirms that it could generate. Alternatively, agent *j* could respond to A3′ with a primitive such as *refuse* or *sorry*, to indicate that it will not perform the revision to its model that A3′ entails.

6.2. Outer message primitives and inner message content

We next turn to the case in which the normative model prohibits a particular illocutionary force applied to a particular proposition, i.e., an outer message primitive in combination with a particular proposition type in the *:content* field. One intuitive example is when agent *i* attempts to inform agent *j* of what agent *j* desires:

Case B

	message B1	(inform *i j* (desire (*j*...)))
	message B2	(not-understood *j i* ((message B1⟩ (not (external desire))))
X	message B3	(not-understood *i j* ((message B2⟩ (external desire))))
?	message B3′	(disconfirm *i j* (not (external desire)))

Informally, agent *i* is telling agent *j* "Here's something you need to know." In message B2, *j*'s *not-understood* reason is that *j* believes that *desire*-propositions which take *j* as an argument cannot be updated by a communication action from another agent. Now, there are two possible responses that *i* could make. Message B3 cannot be sent for the same reasons outlined for message A2 in Case A (and is embodied in the context-sensitive grammar in Table 4). In message B3′, *i* aims to revise *j*'s model about whether the *intend* predicate is external. Will message B3′ be generated? It depends on whether, by agent *i*'s model, the predicate *external* is itself classed as dynamic-external and can be used in propositional message content. Agent *j* will accept message B3 as legal if its own model concurs and thereby process the original message B1; otherwise, it can respond with a *not-understood* or a *refuse* primitive, as in Case A.

Similarly, we can consider whether agent *i* can ask agent *j* to tell agent *i* what agent *i* desires. ("Tell me what you think I need to know."). This is not allowed by our particular grammar for *query-ref* (although a different grammar could allow this). This situation is not limited to modality predicates. The same case holds if agent *i* attempts to query another agent about any property than only agent *i* can know (as bizarre as that might be), such as the status of agent *i*'s own task. Assume that task-23 belongs to agent *i*, but agent *i* asks agent *j* for the task's status:

Case C

message C1	(query-ref *i j* (status task-23))
message C2	(not-understood *j i* ((message C1⟩ (not (external status))))

The grammar in Table 4 prevents agent *j* from formulating an *inform* as a response to C1. There is no allowable response that agent *j* can formulate, except message C2.

6.3. Agent roles for normative communication

The next case adds one additional feature to the preceding one. An outer message primitive combined with some particular propositional content might be legal only for agents that have particular roles or interdependencies. Our intuitive example from Table 1 was "According to my model of you, you cannot know *p*" or "According to my model of our dependencies, you cannot ask me about *p*."

Suppose that *i* proactively sends an *inform* message to agent *j* about the result of task-23. And suppose further that agent *j* does not believe that an *inform* feasibility precondition holds for agent *i*: If only the agent assigned to a particular task can know the results of that task, and *j* does not believe *i* is the agent assigned to task-23, then *j* would believe that an *inform* feasibility precondition ought not to be holding for *i*.

Case D

	message D1	(inform *i j* (result-of (task-23...)))
	message D2	(not-understood *j i* (⟨message D1⟩ (not (agent-for (*i* task-23)))))
?	message D3	(disconfirm *i j* (not (agent-for (*i* task-23))))

It might be quite pragmatic for agent *j* to send the *not-understood* in such a case, if agent *i* were assuming that agent *j* was going to do something of consequence with task-23's result. Message D3 *could* be a legal continuation of this exchange, if the interaction model defines *agent-for* to be a dynamic-external predicate. Now, should agent *j* change its belief about whether agent *i* is responsible for task-23, just because agent *i* says so? That has to be determined by the policies that govern how agents may or must change their interaction models during run time. There could be an axiom that says "*Any agent is the final authority on what tasks it has responsibility for*." or one that states "*There is no reassignment of agents to the* '*password security task*' *during run time*." We note, however, that this puts the burden back on the system designer to ensure that the model correction policy does not yield different outcomes, as these two axioms might.

In this example (and others), *j*'s reason could also simply be the equivalent of "I do not believe you know the result of task-23," without further appeal to a model-based reason. We return to the issue of selecting a reason in Section 6.5.

6.4. Mismatched models for agent roles and dependencies

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:

Case E

	message E1	(inform *i j* (result-of (task-23...)))
	message E2	(not-understood *j i* (⟨message E1⟩ (not (desire (*j*, result-of (task-23))))))
?	message E3	(disconfirm *i j* (desire (*j*, result-of (task-23))))

According to the interaction model, message E1 is not understood because, from *j*'s viewpoint, the contextual relevance conditions for this *inform* are not holding for *i*: *j* does not need to know the result of task-23 and *i* apparently believes otherwise. The *not-understood* reason in message E2, taken as a *disconfirm* from *j* to *i* about *i*'s beliefs, is legal under the interaction model. This causes agent *i* to update its model of agent *j*'s responsibilities. Agent *i* cannot try to revise agent *j*'s belief state with message E3, because by the interaction model, *i* cannot change *j*'s state about what *j* desires. Here, it is agent *i*'s model that is revised to align with *j*'s model.

There could be a different (or additional) reason why agent *j* does not understand message E1. It might not believe that agent *i* has any reason to know the results of task-23, because it believes that agent *i* is not assigned to task-23. This corresponds to case D.

Now, Case E presents an interesting twist, if we consider that message *m* can *become* understandable once message $m+1$ is received. For example, I might first tell you the result of task-23, and *then* inform you that you are assigned a task that needs that result (hence, providing the contextual relevance in the second message). This situation can be problematic if there seems to be no order of messages that can make matters understandable to the receiver (given some particular model), e.g., I cannot inform you that you are the agent for task-23 because you will say you don't know how to do task-23, and I cannot tell you how to do task-23 because you will say you believe you have no reason to know that.[7]

These are still cases of mismatched models between the sender and the receiver. No matter which order is chosen by the sender, a *not-understood* is still warranted as the first message back from the receiver, precisely because it signals mismatching models and its reason gives a hook into the approximate cause of the mismatch. The sending agent can send the follow-up message that (potentially) initiates some model adjustment, e.g., "you are assigned to task-23." This case is logically problematic, only if the underlying axioms that relate agent properties and task properties are circular, e.g., agent *a* is assigned to task *t* iff it desires the resource associated with task *t*. Admittedly, this puts the burden back on the agent designer to think carefully about the underlying inference rules. Assuming just the simple axioms given in Section 4.3, this sort of circularity would not occur. Thus, agent *i* can inform agent *j* that *j* is the assigned agent for some task, so that the first message about results or resources can be understood, within such an adjusted model. From a processing viewpoint, this requires a feature in the conversation layer that holds belief revisions or updates (that come via an *inform* or *disconfirm*) in abeyance, pending the closure of the conversation. This too raises issues in practice, e.g., *x* becomes understandable upon receipt of *y*, which becomes understandable upon receipt of *w*, and so forth, within a given conversation between two agents. In thinking about these more complicated cases, it's important to separate the matter of grounding the conditions for *not-understood* messages in some normative communication model – our main concern here – from the matter of dynamically changing models so that such messages become legal. The second matter requires a decision about just how much real-time model adjustment it is desirable and plausible to have for a given agent-application during run time, and the specification of policies to govern that. We return to this issue below.

As a final case, we consider *not-understood* responses to *query-refs*. Reconsider Case C, in which agent *i* asks agent *j* for information on the status of agent *i*'s own task. Suppose that task-23 were assigned to neither agent *i* nor agent *j*, and agent *i* issues the same query.

Case F
  message F1 (query-ref *i j* (status task-23))
  message F2 (not-understood *j i* (⟨message F2⟩(not (agent-for (*j* task-23)))))

From *j*'s viewpoint, one of the preconditions for *i*'s *query-ref* is not correct, namely agent *j* does not know the status of task-23. The reason is *because* agent *j* is not the assigned to task-23. We can imagine other cases in which a *not-understood* is the normative response to a *query* if some privacy constraints must be upheld, and such constraints must be specified in the interaction model. For example, by some normative communication model, agent *j* can believe that agent *i* has no access to the information it is querying about. Again, exactly whose model ought to carry the day in these dialogues is not a matter of normative communication, but rather policies for normative model alignment.

6.5. From not-understood messages to model realignment

These little *not-understood* conversations underscore the difference between a normative communication model and policies for model alignment. The former defines preconditions and reasons for *not-understood* messages, grounding them as illegal update, revision, or access operations by agents on each other's belief states. The latter defines which of two mismatched models is going to be taken as correct and raises the topic of complex agent reasoning.

Specifying whose model ought to win the day in these *not-understood* message exchanges – and how the losing model is to be revised – can be kept simple or it can become complicated. The simple solution is that the recipient of the *not-understood* just 'accepts' that its model is wrong by incorporating the reason within the *not-understood* message into its belief state, and perhaps tries the same message to a different agent. This is the minimal model adjustment that is required to prevent that very same message from being sent again to the very same agent. The safest way to do this adjustment is to associate the reason with an agent-specific model, e.g., "Agent *j* believes that it is not the agent for task-23" or "Agent *j* believes that predicate *p* is not in such-and-so predicate class." By this approach, a single misprogrammed agent cannot cause widespread model adjustment about, say, what kinds of operations are in fact legal or illegal on each other's internal states.

The more complicated approach allows the sender of the original message to send a follow-up to the *not-understand*, with the intent to change the model of the other agent so that the original message can be accepted as legal. This complicated approach requires policies for how a normative communication model can be modified and a consideration of just how much reasoning it is practical or necessary for an agent system to do with respect to model adjustment. This also bears on the matter of how to select which of several possible reasons to give for a *not-understood*. For example, suppose there were an axiom that only the agent assigned to task *t* knows the status of the task *t*. If there were only one such axiom, there would be only one reason to give when an agent asserts it does not understand a query concerning the status of task *t* (see Scenario F above). But suppose there were a second axiom: *If an agent has the feature "omniscient", then it knows the status of all tasks within the system.* If the goal is to correct agent *i*'s model, then agent *j* might send all possible reasons ("It is false that I am the agent for that task." and "It is false that I am omniscient") without needing to diagnosis which one might be at play for agent *i*.

It is also unclear whether the reason sent with a *not-understood* ought to be the 'deepest' or 'shallowest' reason possible (see case D). If agent *i* asks agent *j* for the result of a task to which *j* is not assigned, *j*'s *not-understood* reason could either be "it is false that I know that result" or "It is false that I am the agent assigned to that task." The latter would reduce the likelihood of other illegal messages going from *i* to *j*, but this would amount to diagnostic reasoning on *j*'s part. The former is the simplest fix to agent *i*'s model of agent *j*; whether *i* ought to do any further reasoning about the root cause is a different matter. If there is going to be 'deep' diagnosis, then either the sender of the *not-understood* will do it, to alleviate reasoning on the part of the receiver, or the receiver of the *not-understood* will do it, to make sense of the shallow reason it receives. If the interaction models are complicated, then ensuring that the model adjustment is correct can require still more message exchanges, and this might not be pragmatic for agent systems. This is a common feature in so-called dialogue repair models of human conversation, which we touch upon briefly below.

7. Related work and themes

If 'agent' is regarded as the next level of abstraction in programming and large-scale system design, then it is natural to ask how the agent paradigm can import elements of software engineering, or vice versa. To this end, various extensions have been proposed to the Unified Modeling Language (UML) to model agent-based interaction via protocols (e.g., [5]). By other accounts [24, 34, 47, 48], what the agent paradigm offers the software system designer is some relief from the need to anticipate all possible interactions among software modules. The vision is to off-load some decision making about unanticipated events during run time to software modules that have some decision making capability and autonomy. This requires a commitment to some kind of agent theory. But with that commitment comes some uncertainty about how agent-like software modules will behave and interact during run time [24]. We believe that some of this uncertainty can be reduced through a clear specification of normative behavior and interaction. In this work, we use normative communication behavior as a reflection of normative domain behavior. Other approaches have focused on frameworks for defining normative domain behavior directly.

The REDUX project management architecture uses the specifications of agent roles and permissions as it monitors the state of a design solution that is being planned and executed [35]. REDUX generates a KQML *sorry* message along with a reason (as per FIPA's *not-understood*) when an agent attempts a domain action for which it does not have the proper role or permission (e.g., changing a deadline). It also monitors how a change to the evolving solution can impact the decisions still to be made by other agents. In these senses, the architecture serves as a central authority for normative domain behavior and to some extent, for promoting more efficient domain behavior among the interacting agents. Petrie et al. [35] make a cogent case for the ways in which a strongly-typed agent language is not merely a convenience for agent message passing, but is also central for system design and debugging, when it is coupled with a strong agent theory and domain behavior model.

Klein and Dellarocus [25] propose a framework in which agents 'register' their normative behavior with a general exception handling module. This general exception detection model serves as a central authority for recognizing deviations from normative behavior. An example exception might be that agent *i* cannot – contrary to the registered expectations – produce a result in time to be used by agent *j* during run-time. It is the exception-handling module (not agent *i* or *j*) that recognizes this event as an error and then recommends some alternative course of action for agent *j*. To employ this approach, agents must share a common language for describing their behavior to the central exception-handling module. Like REDUX, this approach focuses on normative domain behavior directly and makes recognition of illegal interactions the responsibility of a central mechanism. In contrast, our work assumes that attempted communication actions reflect intended domain behavior and we distribute that model of normative communication to all agents. The generation of a *not-understood* can be viewed as throwing an exception.

The GAIA framework [50] supports normative agent behavior principally through the specification of an agent role, defined as a 4-tuple of responsibilities, permissions, activities, and protocols. Responsibilities define functionality, i.e., domain states that an agent brings about or maintains. Permissions correspond to the resources that agents have, in order to execute their responsibilities. These would include access to information (in the same sense as we have used the notion of resource here). Activities correspond to whatever domain actions an agent does privately, without involving other agents. This corresponds loosely to what we denote as methods in the task structure model here, as well as to the results of other internal computations an agent does that are not part of the shared abstraction language. Finally, a role's protocol is much like an interface specification, defined by a sender, a receiver, the inputs used by the sender, and the outputs supplied by the receiver. The GAIA framework takes agents as the starting point for modeling, whereas our approach starts first with modeling the task, the interaction ontology, and the elements that populate an agent's internal state as possible objects or arguments for communication actions. This is the overarching structure, and agent roles (permissions, responsibilities, and communication interfaces) are derived via the axioms that relate task properties to agent properties. The axioms correspond to the adoption of a particular agent theory; a different set of axioms would constitute a different agent theory. So although our approach may appear more task-centric than agent-centric, there is an underlying agent theory that determines certain agent properties, given their assigned responsibilities within a distributed task.

This task-centric approach characterizes much of the distributed AI and multi-agent paradigm, which emerged from the notion that a solution to a task within some resource or time constraints might be done more effectively if that task were decomposed and distributed to independent but interacting problem-solving entities. The TÆMS task modeling system [9, 10] and the GPGP coordination mechanisms [11] evolved from this kind of perspective. The GPGP approach coordinates commitments (and communications about commitments) concerning task results, deadlines, and resources in a multi-agent system. Message passing ensues among agents prior to problem solving, with the aim of developing local scheduling commitments that enable or optimize the interactions that will occur during run time. Wagner et al. [45] note that this message passing itself is a kind of coordination mechanism and they consider how GPGP-like mechanisms can control and coordinate conversations. They propose extending the GPGP framework to encompass conversations about task allocation, global goals, and task progress. Such an extension would require a vocabulary for such conversations, much like the abstract task vocabulary we have adopted here. Our use of a task structure coupled with axioms to derive agent states and intentions is one way to accomplish their proposal for a belief-desire-intention style representation of shared tasks.

In contrast to the task-centric approach, there are more conversation-centric approaches to defining normative communication for multi-agent systems. Agentis [12] uses simplified agent task models coupled with protocol specifications in the domain of registering, providing, and requesting services. Most protocol specifications define normative message exchange simply as a sequence of message types (e.g., *query* followed by *inform*); transitions between conversation states are defined solely by a required outer message type. The Agentis framework allows protocol state transitions to be defined as a particular message type coupled with particular message content. This is important because a protocol or conversation state acts as a proxy for an agent state. The more detailed the transition test can be specified, the more detailed the constraint becomes between agent state transitions. For example, an agent might signal the cancellation of a service, by sending a *decline* message type, with one of several predefined reason types. Those different elaborations of a *decline* put the receiving agent into different states, at least with respect to its subsequent decisions and interactions. This is similar to our use of a restricted predicate set for constructing propositional content, particularly the *processing* propositions. Protocol-like behavior emerges from our task structure and the agent theory that derives agent properties from the tasks to which agents are assigned.

A conversation-centric perspective could also define agents in terms of the conversational interfaces they have with other agents. The GAIA's agent role includes an interface specification to other agents. COOL [2] makes agent responsibilities, activities, and protocols part of the agent programming language. The normative conversation topics and message sequences between two agents are the only ones that such agents are defined and programmed to have. Our work puts these specifications in a declarative representation; the presumption, of course, is that agents are designed to interpret this representation and that they adhere to the same underlying agent theory.

Our task-centric approach also reflects the influence of computational linguistics research on modeling task-directed dialogues, either between two humans or between a human and a software agent. That literature is too vast to review adequately here, but a few key connections are worth acknowledging. A long-standing element of computational and theoretical treatments of task-directed conversation is that domain task intentions give rise to communication intentions, and vice versa (e.g., [30, 31]). This is central to any agent theory that employs even simplistic notions of beliefs, desires, and intentions. Common goals and theoretical constructs such as 'shared plans' are foundational elements for a semantics for cooperation [23] and for implementing complex, mixed initiative human-agent cooperation systems [16, 38]. From our viewpoint, what this literature calls 'mixed initiative dialogue' is what the software engineering literature might call 'strong peer-to-peer communication without a central calling hierarchy.' Although the aims of these two areas are different, their common concern is the resolution of non-observable agent states from observable behavior, e.g., an utterance or a message. More directly related to our concern with the *not-understood* message is the matter of 'dialogue repair.' Within this area, McRoy and Hirst [33] distinguish between the matter of 'misunderstanding' versus 'misconceptions' in a dialogue. The latter corresponds to errors in one agent's model of the domain that is serving as the conversation topic (e.g., the belief that Toronto is the capital of Canada). Such errors are usually recognized when an utterance is not interpretable by the receiving agent's model. In contrast, misunderstandings signal that the receiver's assigned interpretation of an utterance is not the one intended by the speaker; the recognition of this case may take several further message exchanges to detect. The work we report here is closer to the notion of misconceptions, since it centers primarily on errors arising from mismatching interaction models. In [37] a general abduction framework is used to identify and resolve dialogue misunderstandings. Such a general framework could be similarly employed by agents to resolve an incoming message in terms of an underlying normative communication model.

Finally, a critical element of our analysis is the view that communication acts are attempted updates, revisions, or accesses to a portion of an agent's internal state. Our pragmatic approach to defining a model by which particular communications can be classified 'not understandable' or 'without meaning' (i.e., illegal) is a simple instance covered by the more general framework of dynamic semantics [20]. Simply put, dynamic semantics views the meaning of a sentence (here, a message) as the potential for change it can bring about in the information state of the hearer. According to this theory, information states for an agent are a subset of all the 'possibilities', which in turn are defined by a reference system (a function from the referential terms in the language to the domain of discourse) and a possible world. In the ideal case, an agent's information state consists of just one possibility, i.e., one relation between the information that has been exchanged via communication about the real world, and the real world itself. A sentence is 'unacceptable' or without meaning if there is no accessible information state with which it is consistent. Groenendijk, Stokhof, and Veltman [20] claim that 'no hearer will be prepared to update his information state with a sentence if the result would be the absurd state,'' where the absurd state is one in which the possibilities are empty. If agent *j* holds a normative communication model that is inconsistent with agent *i*'s (*inform i j* α), then updating its information state with α would leave it in the absurd state, one with no possibilities left. It is on this occasion that the (attempted) transition between information states represented by a message exchange is illegal from agent *j*'s perspective, and which causes agent *j* to "argue with" agent *i* by sending a *not-understood* message. A nice feature of dynamic semantics is that it accommodates the intuition that the order in which messages (sentences) are exchanged, matters. It also offers the view that an agent engages in message exchange to reduce uncertainty about what the state of the real world is. The essence of these ideas gives theoretical clarity to the role that a normative communication model plays in defining the cases under which a 'not-understood' message *should* be generated. The model serves to completely circumscribe a set of possible worlds within the application domain, including the internal states of all agents in that domain.

8. Discussion

We have presented the notion of a normative communication model that, in its simplest use, defines well-grounded preconditions and reasons for *not-understood* type error messages. The preconditions and reasons are leveraged off the implicit preconditions for the core message types. The error messages signal mismatches among agent models that are driving the agents' domain-level interactions. In its more complicated role, a normative communication model can serve as the foundation for defining policies by which agents might re-align their models during problem-solving. This approach can be employed in any agent system that makes a strong commitment to task-modeling, defines axioms that relate task features to agent features, and uses some kind of propositional representation for that portion of agent beliefs that are exchanged during problem solving. It presumes that agents would have access to the task model, the axioms, and either a partial or a global model of their interacting work. But once a system designer makes the kinds of semantic commitments of the sort we have presented here, and codifies those commitments, then a normative communication model that structures *not-understood* preconditions and reasons follows rather simply. The preconditions of the core message types – however a system designer defines those core message types and preconditions – implicitly define conditions for "understandability" and hence they can be leveraged to define the conditions for "non-understandability."

This work presumes that an agent designer can impart a 'deep model' of joint work and expected interactions to distributed agents, using a common abstract language and a central declarative representation of the model. This is easy to do for a closed system of homogeneous agents, because a single design team can adopt a common agent theory and a common abstract language. But in such a system, one can argue that there is no real need to employ a high level ACL in the first place, for such agents can be programmed to communicate with each other in whatever way the system designer deems appropriate. A good part of the motivation for a high level standardized ACL was to support communication among heterogeneous agents, who most likely have *shallow* models of each other. There are two considerations to raise here. Even if a standard ACL is not employed, the proliferation of strongly-typed agent languages for designing certain types of distributed systems indicates a growing investment in the agent paradigm [34]. That investment in turn requires explicit ways of defining normative communication interactions at an agent level, and hence the investment in message protocols. The second consideration is the matter of just what agents will 'talk about' via the inner language for expressing the *:content* of their messages. So it is not merely a matter of what outer message type is normative to exchange at a particular time, but also what inner message content is normative to exchange at a particular time. Some of the interaction model components we have advocated here (e.g., predicate classes coupled with agent roles) can be used without any kind of shared task model and still provide a kind of context-sensitive semantic check on the legality of a communication action like *inform* or *query*. Viewed another way, the normative communication model of the sort we've described here is a richer representation of protocol-like behavior for moving a distributed task to its solution state.

We have focused on *not-understood* as a canonical error message type that includes a structured reason, as inspired by the FIPA ACL specifications. As we noted, the KQML *error* and *sorry* message types reflect an important distinction between a message that is syntactically incorrect and a message that was interpretable but did not have its intended effect, respectively. FIPA's *refuse* message is similar to *sorry*, provided as a possible response to a request for action that an agent will not take. There can be any number of episode-specific reasons that a request for action message is not honored. These reasons can fall outside a normative communication model and for them, something like *sorry* or *refuse* is warranted instead of a *not-understood* (e.g., agent *i refuses* to schedule a task for agent *j*, not because there are missing elements from the interaction model that would render such a request illegal, but perhaps agent *i* has decided that agent *j*'s requests have lower priority than usual). *Not-understood* is best regarded as a response to make when a received message violates the underlying model that governs message exchange between two agents, with the emphasis on the update, revision, or access operations on the agent state that such message processing entails. An agent may still send *refuse* or *sorry* to signal that it is not going to update or revise its underlying model as part of a *not-understood* message exchange (as per Case A in Section 5); such a message would signal that the agents' policies for model revision were not same. The matter of specifying theoretical and computational models of belief revision and update is a complex matter in itself, entailing assumptions about how contradictions are defined and recognized, whether newly arrived information ought in fact be integrated, and so forth. In this work, we have very modest belief revision and update operations (via *disconfirms* and *informs*) that are directed solely through the interaction model.

Even with these simplifications, several issues remain surrounding if and how agents could use *not-understood* conversations as a means for aligning disparate models of their possible or necessary interactions. These models could diverge during problem solving, through lost messages or through task reallocation during run-time. So it is not unreasonable to consider that simple belief revision might actually be necessary. We touched on several of these matters in our consideration of *not-understood* message exchanges. We are exploring exactly these issues with the aim of evaluating just how out-of-sync agent models can become during run time and still be resolved through simple policies that govern model revision.

Another assumption of our approach is that agents have some model of the belief states of other agents. Maintaining models of mutual belief presents difficult theoretical and practical problems [15] and it is easy to get mired in some kind of infinite regress in generating reasons for *not-understood*, i.e., "I do not understand message *m* because I thought that you thought that I thought that you thought..." Here, we have simplified matters greatly by restricting an agent's internal belief state to simple propositional elements and implicitly allowing at most a three-level model of mutual belief (I believe that you believe that I believe) based on the shared task model and run-time information exchange. It remains to be seen how well these assumptions hold up for more complex interactions and agent systems. Finally, the axioms that we employed could be represented in agent markup language such as DAML [8]. This would lift elements of the agent theory out of its internal hard coding into a declarative representation as part of the overarching model. Our current work makes such axioms part of a declarative specification of constraints.

Agent message types have intended effects that are, in their first instance, manipulations, updates, or revisions of the mental (informational) state of the receiving agent. A canonical *not-understood* message constitutes an agent's recognition that an illegal action has been attempted on its internal state. These illegal actions are defined in terms of a normative communication model, which references agent properties derived from a task model. Our approach can be employed in any agent system that uses a structured ACL, because the core elements of a normative communication model are first leveraged off the preconditions that an ACL designer associates with each message type. Other elements of our approach, such a predicate classes for constraining message content in the context of particular senders and receivers, apply to any system in which propositional beliefs constitute part of the agent state. When these constraints are combined with agent roles, the interaction model can become even richer and more complex (e.g., *some* agents may make *some* kinds of operations on the internal state of other agents, depending on their respective roles). Such elements define a principled approach to constraining message content, message exchange patterns, and thereby a set of error conditions for agent communication. By focusing on communication as actions upon internal agent state, the model that defines the set of *not-understood* messages also constrains the possible replies to *not-understood* messages and the allowable belief revisions and updates to agent models.

Acknowledgments

This work was supported by a Natural Sciences and Engineering Research Council Discovery Grant to R. Elio. We thank Jeff Pelletier, Eleni Stroulia, and two anonymous reviewers for their suggestions, questions, and commentary about this work and its presentation.

Notes

1. We use the acronym "ACL" to mean agent communication language, in the generic sense.. To denote FIPA's specific proposal for an ACL, we use "FIPA" or "FIPA's ACL."
2. To put this in more linguistic terms, such a model would define the cases in which it is proper for a speaker to apply a particular illocutionary force to some propositional content, in conversation with a particular hearer.
3. A normative communication model is silent on the matter of ensuring that the receiver's internal state is consistent with α. And as we noted earlier, a different semantics for *inform* could specify a different post condition or intended effect.
4. The feasibility preconditions for *disconfirm* are that the sender believes $\tilde{}\alpha$ and believes that the receiver is either uncertain about the truth of α or believes α. The intended rational effects are that the receiver believes $\tilde{}\alpha$.
5. For example, given a modal logic with a possibility operator $\lozenge$, we could express this constraint as follows: if agent(a) and agent(b) and task(t) then $\lozenge$ intend(a, (do (b, t))) and $\lozenge$ intend(b, (do (b, t))) and $\lozenge$ inform(b, a, (intend (b, do(b, t)))) and ~ $\lozenge$ inform(a, b, (intend (b, do(b, t)))).
6. *Query-ref* is a FIPA message primitive used to obtain a variable binding.
7. We thank one of our reviewers for raising this interesting case.

References

1. J. L. Austin, *How to Do Things with Words*, Harvard University Press: Cambridge, MA, 1962.
2. M. Barbuceanu and M. S. Fox, "COOL: A language for describing coordination in multi agent systems," in *Proceedings of the First International Conference on Multi-Agent Systems*, Holden-Day: San Francisco, California, pp. 17–24, 1995.
3. M. E. Bratman, "What is intention?" in P. R. Cohen, J. L. Morgan, and M. E. Pollack, (eds.), *Intentions in Communication*, MIT Press: Cambridge, Massachusetts, pp. 15–32, 1990.
4. P. Bretier and M. D. Sadek, "A rational agent as the kernel of a cooperative spoken dialogue system," in J. P. Müller, M. Wooldridge, and N. R. Jennings, (eds.), *Intelligent Agents III (LNAI Vol. 1193)*, Springer-Verlag: Berlin, pp. 189–204, 1997.
5. B. Bryson, J. Müller, and J. Odell, "An extension of UML by protocols for multiagent interaction," in *International Conference on Multiagent Systems*, Hilger: Boston, Massachusetts, pp. 207–214, 2000.
6. P. R. Cohen and H. J. Levesque, "Intention is choice with commitment," *Art. Intell.* vol. 42, no. 2–3, pp. 213–261, 1990.
7. P. R. Cohen and C. R. Perrault, "Elements of a plan-based theory of speech acts," Cognitive Sci. vol. 3, pp. 177–212, 1979.
8. DAML, "Agent Markup Language," www.daml.org <http://www.daml.org>., 2003.
9. K. Decker, "Environment centered analysis and design of coordination mechanisms," Technical Report 1995-069, Department of Computer Science, University of Massachusetts, Amherst, Massachusetts, 1995.
10. K. Decker and V. Lesser, "Quantitative modeling of complex computational task environments," in *AAAI-93 – Proceedings of the Eleventh National Conference on Artificial Intelligence*, Brookings Institute Washington, DC, pp. 217–224, 1993.
11. K. Decker and V. Lesser, "Designing a family of coordination algorithms," in *Proceedings of the First International Conference on Multi-agent Systems*, Holden-Day: San Francisco, California, pp. 73–80, 1995.
12. M. d'Inverno, D. Kinny, and M. Luck, "Interaction protocols in Agentis," in *Proceedings of the Third International Conference on Multi-agent Systems (ICMAS-98)*, Universiteires de Press: Paris, France, pp. 112–119, 1998.
13. R. Elio and A. Haddadi, "On abstract models and conversation policies," in F. Dignum and M. Greaves, (eds.), *Issues in Agent Communication (LNAI 1916)*, Springer-Verlag: Berlin, pp. 301–313, 2000.

14. R. Elio, A. Haddadi, and A. Singh, "Task models, intentions, and agent communication," in *Proceedings of the Pacific Rim Conference on AI (LNAI 1886)*. CSIRO: Melbourne, Australia. pp. 394–403, 2000.
15. R. Fagin, J. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about Knowledge*, MIT Press: Cambridge, Massachusetts, 1995.
16. G. Ferguson, J. Allen, and B. Miller, "Trains-95: Towards a mixed-initiative planning assistant," in *Proceedings of the Third Conference on Artificial Intelligence Planning Systems*, Scottish Academic Press: Edinburgh, Scotland, pp. 70–77, 1996.
17. FIPA, "Agent Communicative Act Library Specification," www.fipa.org <http://www.fipa.org>, 2003.
18. M. R. Genesereth and S. P. Ketchpel, "Software agents," Commun. *ACM*, vol. 37, pp. 48–53. 1994.
19. M. Greaves, H. Holmbeck, and J. Bradshaw, "What is a conversation policy?" in F. Dignum and M. Greaves, (eds.), *Issues in Agent Communication (LNAI 1916)*, Springer-Verlag: Berlin, pp. 118–131, 2000.
20. J. Groenendijk, M. Stokhof, and F. Veltman, "Coherence and modality," in S. Lappin, *The handbook of Contemporary Semantic Theory*, Blackwell: Oxford, pp. 179–213, 1996.
21. B. Grosof and Y. Labrou, "An Approach to using XML and a rule-based content language with an agent communication language," in F. Dignum and M. Greaves, (eds.), *Issues in Agent Communication (LNAI 1916)*, Springer-Verlag: Berlin, pp. 96–117, 2000.
22. B. Grosz and C. Sidner, "Attention, intentions and the structure of discourse," *Comput. Linguistics*, vol. 12, pp. 175–204, 1986.
23. B. J. Grosz and S. Kraus, "Collaborative plans for complex group action," *Artif. Intell.*, vol. 86, pp. 269–357, 1996.
24. N. R. Jennings, "On agent-based software engineering," *Artif. Intell.*, 117, pp. 277–296, 2000.
25. M. Klein and C. Dellarocas, "Exception handling in agent systems," in *Proceedings of the Third International Conference on Autonomous Agents (Agents '99)*, Seattle, Washington, pp. 62–68, 1999.
26. Y. Labrou and T. Finin, "A semantics approach for KQML: A general purpose communication language for software agents," in *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, Gaithersburg, Maryland, 1994.
27. Y. Labrou and T. Finin, "A proposal for a new KQML specification," Technical Report #CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland, Baltimore, Maryland, 1997.
28. Y. Labrou and T. Finin, "Semantics and conversations for an agent communication language," in M. Huhns and M. Singh (eds.), *Readings in Agents*, Morgan Kaufmann, Holden-Day: San Francisco, California, pp. 235–242, 1998.
29. Y. Labrou, T. Finin, and Y. Peng, "The current landscape of agent communication languages," *IEEE Intell. Sys.*, vol. 14, pp. 45–52, 1999.
30. L. Lambert and S. Carberry, "A tripartite plan-based model of dialogue," in *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, University of California Press: Berkeley, California, pp. 47–54, 1991.
31. D. Litman and J. Allen, "Discourse processing and commonsense plans," in P. R. Cohen, J. L. Morgan, and M. E. Pollack, (eds.), *Intentions in Communication*,MIT Press: Cambridge, Massachusetts, pp. 365–388, 1990.
32. K. Lochbaum, "The use of knowledge preconditions in language processing," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Lidee: Montreal, Canada, pp. 1260–1266, 1995.
33. S. W. McRoy and G. Hirst, "The repair of speech act misunderstandings by abductive inference," *Comput. Linguistics*, vol. 21, pp. 435–478, 1995.
34. C. Petrie, "Agent-based software engineering," in *Agent-Oriented Software Engineering, First International Workshop, AOSE*, Limerick, Ireland, 2001.
35. C. Petrie, S. Goldman, and A. Raquet, "Agent-based project management," in M. Wooldridge and. M. M. Veloso, *Artificial Intelligence Today: Recent Trends and Developments (LNAI* #1600), Springer-Verlag: Berlin, pp. 339–363, 1999.

36. J. Pitt and A. Mamdani, "Communication protocols in multi-agent systems," in F. Dignum and M. Greaves, *Issues in Agent Communication (LNAI* #1916), Springer-Verlag: Berlin, pp. 160–177, 2000.
37. D. Poole, R. Goebel, and R. Aleliunas, "Theorist: A logical reasoning system for defaults and diagnosis," in N. Cercone and G. McCalla, *The Knowledge Frontier: Essays in the Representation of Knowledge*, Springer-Verlag: New York, pp. 331–352, 1987.
38. C. Rich, and C. L. Sidner, "COLLAGEN: When agents collaborate with people," in *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, California, pp. 284–291, 1997.
39. M. D. Sadek, "A study in the logic of intention," in *Proceedings of the Third Conference on Principles of Knowledge Representation and Reasoning*, San Francisco, California, pp. 462–473, 1992.
40. J. R. Searle, "What is a speech act?" in M. Black, *Philosophy in America*, Cornell University Press, New York, pp. 221–239, 1965.
41. Y. Shoham, "Agent oriented programming," *Artifi. Intell.*, vol. 60, pp. 51–92, 1993.
42. Q. Situ and E. Stroulia, "Task-structure based mediation: The travel-planning assistant example," in *The Proceedings of the 13th Canadian Conference on Artificial Intelligence*, Lidee: Montréal, Québec, Canada, pp. 400–410, 2000.
43. E. Stroulia and A. Goel, "Functional representation and reasoning in reflective systems," in J. Appl. Intell., vol. 9, pp. 101–124, 1995.
44. F. Veltman, "Defaults in update semantics," *J. Philos. Logic*, vol. 25, pp. 221–261, 1996.
45. T. Wagner, B. Benyo, V. Lesser, and P. Xuan, "Investigating interactions between agent conversations and agent control components," in F. Dignum and M. Greaves, *Issues in Agent Communication (LNAI 1916)*, Springer-Verlag: Berlin, pp. 314–330, 2000.
46. T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing: New Jersey, 1986.
47. M. Wooldridge, "Agents and software engineering," *Artifi. Intell.*, vol. 11, no. 3: pp. 31–37, 1998.
48. M. Wooldridge and N.R. Jennings, "Software engineering with agents: Pitfalls and pratfalls," *IEEE Internet Computing*, vol. 3, pp. 20–27, 1999.
49. M. Wooldridge and N.R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Eng. Review*, vol. 10, pp. 115–152, 1995.
50. M. Wooldridge, N. J. Jennings, and D. Kinny, "The Gaia Methodology for Agent-oriented analysis and design," *J. Autonomous Agents and Multi-agent Sys.* vol. 3, pp. 285–312, 2000.