

Facilitating Open Communication in Agent Systems: the InfoSleuth Infrastructure

Marian H. Nodine* and Amy Unruh*,†

* MCC, 3500 West Balcones Center Drive, Austin, TX 78759
{nodine, unruh}@mcc.com

† Systems and Software Lab, DSPS RDC, Texas Instruments
PO Box 655303, MS 8378, Dallas, TX, 75265

Abstract. This paper addresses issues in developing open multiagent systems, in which it is easy to expand the functionality by adding new agents with new capabilities, and which facilitate interoperability with other agent systems. We argue that an open multiagent system should define the following support elements for agent communication:

1. A common set of *speech acts* to define the types of messages that an agent might send to another agent.
2. A common *service ontology* by which the agents can describe their capabilities to each other, and reason about which agents have the capabilities needed to execute specific tasks.
3. A common set of *prescriptive conversation policies* to define the acceptable exchanges of messages between agents.

In addition to the above, we also discuss the utility of having a matchmaking agent that can reason over agent capabilities to recommend agents for specific tasks, where the capabilities and requirements are defined using a common service ontology. This ensures that the semantics of matching agent capabilities to task requirements remains the same across the multiagent system.

1 Introduction

Autonomous, intelligent agents are currently being developed in many different application areas. In a multiagent system, each agent executes specific types of tasks, and serves a specific purpose. No agent does an entire job. Rather, it does what it can, then delegates tasks to other agents. No one agent has control over how a task is executed or who executes it, other than controlling the execution of its own specific subtasks.

Because of their modular, cooperative and distributed nature, it should be easy to expand the functionality of a multiagent system by adding new agents that have different capabilities. Ideally, these agents should be able to be added by constructing them independently of the multiagent system, then providing some means by which the multiagent system can find out about the agent and its capabilities. Another attractive option with respect to agent systems is to allow different but related multiagent systems, developed independently, to interoperate.

An open agent architecture is designed to facilitate the addition of new agents to a multiagent system, and to facilitate the interoperation of related multiagent systems. In

an open agent system, the agents operate within a common infrastructure that governs how the agents advertise their capabilities and services, how they select which agent is appropriate for executing a given task at a given time, and how they communicate with one another during the execution of a task. This means providing all agents not only with a common understanding of specific messages and message types, but also a common understanding of the dialogs that can occur between pairs of agents, and among subgroups of agents that were specifically designed to work together. Furthermore, if agents are self-describing over some common service ontology that can be assimilated by other agents, this facilitates the integration of agents with new capabilities. In this paper, we specifically address issues involving inter-agent communication that impact the openness of a given multiagent system.

2 Keys to Open Communication

In a multiagent system, the need for agents to communicate is obvious. Several loose standards for agent communication have been proposed (e.g., KQML [6], FIPA [19], and even in some cases, CORBA). Furthermore, at least in the information integration area, several multiagent systems with sufficiently similar communication paradigms have evolved to the point where it is reasonable to start to attempt interoperation. Our agent system, InfoSleuth, is one of these. InfoSleuth supports a dynamic set of agents, spread out across a network, that collaborate to do information retrieval and integration.

Our experience so far has led us to appreciate the need for standard, open agent architectures to allow for the easy integration of new agents into an existing multiagent system, and to allow for the easy interoperation of related multiagent systems. Many multiagent systems are “closed”, either because they communicate using their own protocols, or they assume that each agent has specific, built-in knowledge of most other agents in their system. In this paper, we address several overall requirements that openness imposes on the architecture of a multiagent system. We will use our experiences with InfoSleuth to illustrate these issues.

We propose that keys to communication in open systems include a shared set of *speech acts* (“verbs”) to provide a structure to the discourse among agents, a common service ontology (a shared set of “nouns” and “adjectives”) which provides a dictionary of meta-information about agent capabilities, and a shared set of prescriptive conversation policies to provide a structure for basic agent dialog. The greater the extent to which this shared information is supported, the easier agent integration and interoperation will be. Also, these requirements are separate from the communication issues involved in the use of a shared domain ontology and domain communication language to execute domain-specific tasks.

2.1 Speech acts, or “verbs”

The concept of *speech acts*, or *illocutionary acts*, has grown out of philosophy and linguistics research [4]. These actions include requesting, promising, offering, acknowledging, asserting, etc. It is suggested that human utterances are the observable byproduct of such actions.

Speech act theory has been proposed recently as the foundation for inter-agent communication. The use of a standard set of speech acts in open systems provides a structure to agent discourse in which the intended meaning of the content of the speech act (what is being said) can be interpreted more easily, and provides a semantic structure to the messages intuitive to the human users of a system.

It has been observed [14] that the speech acts in existing agent systems fall primarily into two general categories: *requests* and *assertions*. This suggests — and has been borne out by our own observations — that there is a strong overlap in the speech acts required by many agent systems, and that a small comprehensive set would be sufficient for many multiagent systems.

Currently, there are several efforts towards defining this standard, comprehensive set of speech acts. Unfortunately, these efforts have not reached the point of agreement, and therefore a single illocutionary act is often represented differently by the different multiagent systems. This impedes interoperability because the systems may not understand each others' speech acts. Worse, the different systems may interpret the same “standard” speech act in different ways (e.g., does TELL imply that the contents must be remembered by the agent being told, or not? Does it imply that the contents must be true or not?). The more complete the effort for standardizing speech acts and the greater the adherence to the standard, the more “open” the agent system will be.

2.2 Service ontologies, or “nouns” and “adjectives”

One fundamental aspect of openness is that an agent should not have *a priori* knowledge of (most) other agents in the system. Rather, if a task requires an agent with a specific capability, it must be able to locate such an agent. In a fairly stable system, perhaps the agent information can then be cached; however, in a fully flexible system where agents may enter and leave the multiagent system, this location information may need to be generated frequently.

For this to occur, a common means of representing and exchanging service and capability information is important. The exchange of this information can be done either in the form of an advertisement or in answer to a query. This advertisement or query must be specified in terms of a predefined set of common semantic concepts, which we call a “service ontology”. The service ontology may contain the “nouns” indicating *what* it can do, and the “adjectives” limiting or further describing these capabilities. To our knowledge, no such common service ontology has been defined.

Once the capabilities of an agent are expressed in terms of this common service ontology, the current requirements of a task may be formulated in terms of the same ontology, making it possible to deduce whether or not the agent is a good match for the task. Thus, a multiagent system's matchmaking and brokering capabilities contribute to its openness.

2.3 Prescriptive conversations

A set of standard messages representing the available speech acts, and a common service ontology, serve as a basis for very simple communication among agents. However, messages do not get sent in isolation; rather, there are often ongoing dialogs among two

or more agents. Within a dialog, the interpretation of an individual message may depend on the *context* of the dialog in which they are participating. Take, for example, a SUBSCRIBE message. This message makes sense if it is received out of the blue by some agent that has the knowledge requested, and has the capability of returning and noticing changes in that knowledge. However, if this message is received in reply to a transmitted ASK-ONE message, it should be an error.

A “conversation” is a partially-ordered set of messages transmitted among a fixed set of agents, all of which relate conceptually to an initiating speech act. A “conversation policy” is the formal and deterministic specification of the ordering of speech acts exchanged between two agents during a conversation. A “prescriptive conversation policy” is one that is defined *a priori*, as opposed to one that evolves during the course of a specific conversation.

Open agent communication requires that any two agents that communicate share a common set of conversation policies. When an agent initiates a conversation or receives an incoming conversation request, it needs to know what types of replies are appropriate. If the conversation can be interrupted (e.g., if the remote agent is sending a stream of responses to the SUBSCRIBE), then each agent must know how and when it can correctly interrupt the conversation.

We believe that all agents in an open system must support a default set of prescriptive conversation policies. These policies must be prescriptive because we can make no assumptions that two agents will reason about conversational responses in the same way, nor can we assume that all agents have the capability to do this kind of reasoning. Clearly, the more such policies that are shared among a group of agents, the richer the agent interaction will be.

3 Overview of InfoSleuth

The InfoSleuth project at MCC [2, 10, 16, 17] is investigating and developing technologies to support a robust and extensible agent architecture for heterogeneous information gathering and discovery in a dynamic environment. InfoSleuth views an information source at the level of its relevant semantic concepts, thus preserving the autonomy of its data. Information requests to InfoSleuth are specified generically, independent of the structure, location, or even existence of the requested information. InfoSleuth filters these requests, specified at the semantic level, flexibly matching them to the information resources that are relevant at the time the request is processed.

Key components supporting our approach include:

- the representation and use of common ontologies with which all agents communicate;
- the use of a *broker* agent with deductive reasoning capabilities, to which all other agents advertise their capabilities and to which the agents then can formulate queries about desired requirements;
- the use of task scenarios to support different types of high-level queries; and
- techniques for query decomposition over dynamically located relevant resources.

In the remainder of this paper, we discuss each key to open communication, proposed in Section 2, in the context of the InfoSleuth implementation. Section 7 then discusses related issues and concludes.

4 Speech Acts and KQML

The use of a standard set of speech acts in open systems provides a structure to agent discourse in which the intended meaning of the *content* of the speech act (*what* is being said) can be more easily interpreted. In InfoSleuth, we have chosen to build on KQML to define the set of speech acts used in our system. KQML [11, 6] is an effort to define standard useful set of speech acts, or *performatives*, that agents can use to exchange information; as well as the semantics behind the performatives. The performatives each have a number of fields, or parameters associated with them — in addition to the *content* of the performative, other *contextual* parameters specify e.g., the language and ontology of the message. Routing parameters specify the sender and receiver.

By factoring the contextual and routing information out of the content of the message in a standard manner, the goal is that agents receiving a message will be able to analyze and possibly route the message, independent of whether or not they can understand the content itself. An important research issue is the question of how *much* content information should be pushed to the level of the performative. As an example, consider the general category of *requesting* performatives. In KQML, the SUBSCRIBE and ASK-* performatives both fall into this category. Both ask for information, but differ in what they expect in return. The semantics of this expectation could conceivably be pushed down to the content of the message (which could then be wrapped in a more general REQUEST performative). However, offloading this semantics to the performative level, and making an explicit distinction between a SUBSCRIBE and an ASK, is useful because it enables the agents to obtain important information about the request without having to parse the content. On the other hand, too much complexity at the performative level, especially with those performatives that are not commonly used, will inhibit its utility as an open standard.

Though KQML is not yet stable as a standard, we believe it is a useful vehicle to drive community-wide research in defining a commonly used set of performatives, as well as research into what the structure and fields of those performatives should be. In practice, the InfoSleuth group has found the need to use only a subset of the existing performatives, while finding it useful to define new performatives as well. One performative in particular, ADVERTISE, is particularly pertinent to the exchange of capability information in open systems, and we discuss it in more detail in Section 5.

5 Agent Services and Capabilities

In an open system, no a priori knowledge of most other agents is assumed. For this reason, a common means of exchanging service and capability information is crucial to open communication: for agents to effectively and robustly interoperate in a dynamic

system, there needs to be some way for the agents to identify each other's roles and capabilities. The greater the extent to which this is achievable, the more effective the agents' communication and interoperation will be.

In this section, we discuss several aspects of use of capability information in an open system: the content of the service vocabulary, the representation of the service information, and the necessity for a means to formulate requirements about agent capabilities as well as the existence of semantic matchmaking functionality to reason about the requirements.

5.1 Service Ontology as Support for Open Communication

Service and capability information needs to be communicated between agents using a common *ontology* and shared representation rich enough to support it. The contents of a service ontology will contain both domain-independent and domain-dependent components, where the domain-independent portion describes information needed for any type of agent task, such as availability of an agent, or the performatives the agent understands. The domain-dependent portion may describe aspects of an agent's functionality relevant to a certain class of tasks (such as information gathering or design). An agent in an open system must be able to understand both a standardized common ontology describing domain-independent capability information, as well as the ontology describing capability information for its task domain¹.

The InfoSleuth service ontology is driven by the needs of agents in *information-gathering* domains. It includes the following information, which we believe describes a minimum set of concepts for effective agent interoperation in an information-gathering domain. Currently, the domain-independent capabilities are not yet factored out. Research is ongoing to determine what extensions to the ontology are useful, and to identify the subset of the following which should be understood by agents engaged in other types of tasks as well.

- An agent's unique ID
- The type of agent functionality (e.g., *user agent*, *resource agent*, etc., in InfoSleuth)
- Supported ontologies
- Supported content languages
- Supported conversations
- Access cost
- Access time
- Reliability
- Transactional capabilities
- Notification capabilities
- Meta-information about the classes of objects an agent has information about, with respect to an ontology (e.g., schema information).
- Constraints on the *contents* of the objects an agent knows about, with respect to an ontology.

¹ By "task domain", we do not refer to an application domain, which for InfoSleuth may range from healthcare administration to wafer processing. Rather, we refer to the class of activity, such as information-gathering and discovery, towards which the agents are employed.

In [5], a strikingly similar set of capabilities is described. This suggests that our eventual goal of an inter-system service ontology is feasible.

5.2 Representation of Service Information

Agent service information has the potential to be represented in a number of different ways, with respect to the semantic level at which it is encoded. Specifically at issue is the question of what service information should be encoded at the speech-act level, and what should be encoded as the *content* of speech acts. Existing approaches range along this spectrum.

The KQML specification [11] suggests that an ADVERTISE performative should be used to exchange service information, where embedded in an ADVERTISE is a performative (such as an ASK-ALL) that the agent supports. Thus, KQML pushes the capability description entirely into the speech-act syntax. However, we found that the KQML ADVERTISE, with an embedded performative, was not sufficient as a mechanism for exchange of the capability information in the InfoSleuth ontology. As a simple example, many of our agents can answer the same ASK-ALL query. However, each type of agent can have different capabilities with respect to that query. Some agents (*resource* agents) map a query directly to a resource. Some agents (*multi-resource query* agents) perform multi-resource decomposition and joins over a query. And some agents (*task execution* agents) identify and execute a high-level scenario appropriate to answering a query (during which other agents may be identified and queried). Each agent would appear equivalent if another agent had to discriminate based only on the information about the queries they can answer. However, functionally, they are very different, and only one will be appropriate in a given context. Other agents must clearly be able to make this distinction for communication to work in an open environment where a priori information about such functionality is not necessarily available.

One way to make this functionality distinction would be to extend the performatives to make them more expressive. However, in InfoSleuth, we take the approach that speech act communication protocols should not be overloaded to express capability information. Rather, expressions of service capability, and queries on those capabilities, should both be encoded in the content of a performative, consistently with the way that information about and queries on other resources in the system are expressed.

In InfoSleuth, all of the capabilities listed in Section 5.1 are expressed via the content of messages exchanged between agents, using a shared interchange language. The ontological representation and concepts must be standard ones shared by all agents, either directly (as is the case for InfoSleuth) or via a gateway agent that provides interchange translation services.

In [5], the basic insufficiency of the ADVERTISE performative as specified is also recognized, and a similar approach is taken. However, here the representation of the ontology is deliberately tied to KQML. A SERVICE expression is generated for the content of the ADVERTISE, where the representation of the service expression is motivated by a key feature of KQML — that the parameters of the expression may be understood by most other agents even if the content is not. Thus, the SERVICE expression uses KQML syntax and shares some of its field names (such as `:ontology` and `:language`) with

KQML. However, their approach does not alleviate the need for a common service ontology, as evidenced by the fact that they define other classes of information, both “basic” and that specific to “service”, or functionality type. Thus, they mingle service-type information with other standard KQML parameters.

5.3 Name Services

Agent name service (ANS) information and the capability information of Section 5.1 are at different semantic levels in an agent architecture [7, 13]. The name service information is needed to facilitate basic communication, and thus is arguably needed by all agents as a underlying layer upon which the semantic activities needed for open communication can take place.

The name service information should not be confounded with the semantic content of a message; its representation should be factored out, so that implementation of the two layers can be independent. In the community of researchers using speech acts and KQML, at least two different representational approaches exist to do this.

Name Service Information in the Speech-Act Wrapper. The current KQML specification includes routing information — `:sender` and `:receiver` parameters — in its wrapper. Because the name service information is very basic, it is feasible that it can be standardized at the speech-act level, in contrast to the more semantically ambiguous agent capability information. For name service information, KQML then serves as the interchange language. An interchange name-service content language is not needed, making it more likely that disparate systems can communicate. This is the approach that InfoSleuth currently uses. Petrie [13] proposes an additional set of KQML “administrative” performatives that further support and extend ANS and routing activities and add additional service specifications at the KQML level.

Name Service Information as a Layer under KQML. A second approach factors out the name service information from the speech act information, allowing the address information to be handled independently by an underlying transport layer. “Classic” KQML [8, 9] facilitates this, by factoring out the routing information from the contextual and content information about the speech act being sent. The benefits of this representation are illustrated, for example, in the exchange of *nomadic transactions*, where the content of a message may be a set of nested performatives to be passed along as a script; the factorization of the address information makes the management of the nested script cleaner. This representation also facilitates using other message routing services, such as CORBA. In this case, the routing information can be omitted, leaving only the semantic information explicitly represented in the message.

5.4 Use of Capability Information

To make use of capability information, agents in an open system must be provided with three supporting functionalities. First, they must be able to exchange the capability information with each other. Second, they need to be able to formulate requirements on

those capabilities. Finally, they must have a means for *semantic matchmaking*, or reasoning about whether expressed capabilities match a set of requirements.

In InfoSleuth, all of these activities are supported in the same language: LDL++ [18], a logical deduction language with a semantics similar to Prolog, but which supports transparent access to external databases as well as its own fact base. In InfoSleuth (and similarly in several other agent systems, e.g. [12]), *broker* agents are created to specifically provide semantic matchmaking capabilities. In InfoSleuth, the broker agents use a set of LDL++ deductive rules to support inferences about whether an expression of requirements matches a set of advertised capabilities. All other agents in the system express their capabilities to a broker using a TELL performative² (with an LDL++ expression as the content). The broker then accepts matchmaking queries from all agents. These queries to the broker use the ASK-* performatives, consistent with the way agents query other (perhaps domain-specific) resources in the system.

Pragmatically, the use of broker agents means that other agents in the system do not have to implement deductive reasoning capabilities themselves, and do not have to poll a large number of agents to find one which matches the desired capabilities. As long as the agents know the service ontology, and can formulate a query expressing a set of requirements, they can rely on brokers to identify agents which match the requirements.

We suggest that the use of brokers or matchmakers is not only expedient, but a key to the successful operation of an open system. If all other agents work with the same broker(s) to match capabilities, then they are assured of using the same semantics with respect to what it means to match service capabilities. For example, it is possible to use either the closed-world assumption or the open-world assumption when identifying resources that match a certain requirement. It is difficult to enforce that new agents entering a multiagent system are employing matching semantics consistent with what is already in place, and to debug system behavior when they are not. However, if the new agents must use the existing broker(s) in the system to identify resources, consistency of matchmaking semantics is enforced. (The semantic consistency problem remains if new brokers are added to an existing multiagent system, but is reduced in scale).

6 Conversation Policies

While a set of standard performatives and a shared vocabulary for describing agent capabilities provide the foundation for agents in an open system to identify and exchange messages with each other, these capabilities alone are not enough to support open agent communication. The agents need to be able to share a common conversational structure as well. To support open communication, an agent needs to know as best as possible how to interpret responses to performatives it sends out, and needs to know how to respond to incoming performatives such that the receiving agent will interpret the response in the intended way. For example, an agent needs to know that a SORRY is an appropriate response to its ASK-ALL. Conversely, an agent must know what is “expected” of it upon, e.g., receipt of an incoming TELL — is a reply expected by the sending agent? If

² In [5], ADVERTISE is used for this purpose, as a *commissive* speech act specifically different from TELL. In InfoSleuth, no distinction with respect to commissive semantics is made.

these interpretations are not well-specified across a system of agents, then the behavior of new agents may not be stable.

6.1 Need for Prescriptive Conversations

The solution we take in InfoSleuth to support open conversations, is to impose the use of a system-wide *prescriptive* conversation policy. We first discuss the difference between *emergent* and prescriptive conversations in a system of agents. Emergent conversations are those in which the agents are not following specific external conversational policies; but rather where the performatives they use are determined by their internal functionality. These actions may be (but are not necessarily) driven by the agent’s semantic understanding of the discourse taking place. If agents share the same semantics of discourse, a coherent emergent conversational structure may be generated, e.g. [11, 14].

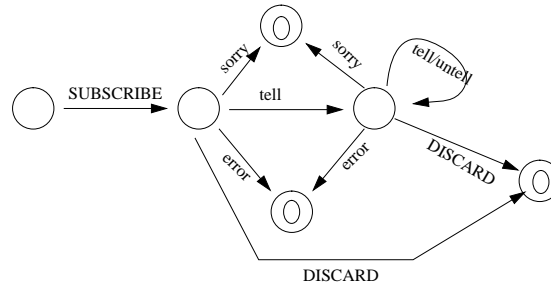
However, in an open system, one cannot make assumptions about the agents’ common understanding of a semantics for discourse. Nor are semantic mismatches easy to debug. Instead, agents need to both expect and provide similar responses in similar contexts. To support interoperation, it becomes necessary to impose structure on at least some functional subset of the agents’ conversations to ensure that they are all following the same conversational rules, or policies. The imposition of structure creates a prescriptive conversational policy.

More specifically, we believe that all agents in an open system must support and enforce a default shared set of prescriptive conversational policies. The larger the shared set, the greater the well-defined agent interaction that can occur. Note that the requirement for system-wide default conversational policies does not preclude situations where some subset of the agents in a system support a larger range of conversations. Note also that the use of shared prescriptive conversations does not guarantee semantically consistent actions upon receipt of a message. For example, an agent may “know” that a REPLY needs to follow an ASK-*, but in an open system there is no guarantee (at the requesting agent’s end) that the REPLY contains what *it* would consider a semantically reasonable response to its question. Nevertheless, we believe that the use of conversation protocols facilitates consistency in an open system by providing the agent with a definition of which message types it can expect and when; and by providing an implicit semantics for the content of those messages.

6.2 Definition of Conversation Structure

The first step in creating a conversation policy is to define the set of allowable conversations between any two agents in the multiagent system. Any given agent can initiate some subset of these conversations, and can respond to some (possibly different) subset. These subsets can then be used to determine *whether* and *how* any two specific agents can interact with each other.

Because in an open agent architecture, agents should have no real a priori knowledge of each other, conversation types should be uniquely identifiable by the types and/or parameters of the messages that initiate the conversation. For example, in an ASK-ALL message, the knowledge of its type (ASK-ALL) is sufficient to determine what the response should be. Most multiagent systems handle ASK-ALL consistently. However,



A SUBSCRIBE contains an embedded ASK-* performative.

Fig. 1. A finite-state model describing a SUBSCRIBE conversation. The messages from the initiating agent are in uppercase; the messages from the responding agent are in lowercase.

some agents acknowledge a TELL message while others do not. It is inappropriate to decide whether or not to expect a response on the basis of which agent you are sending the TELL to, if you wish to maintain an open agent system.

In InfoSleuth, we have used finite-state machines and (in one case) pushdown automata to specify the allowable pairwise conversations between agents in our system. Other work has used a similar approach [3, 11, 15, 19]. Figure 1 shows an example of a conversation type. The transition from the start states in all models correspond to the sending of the conversation-initiating message. The remaining transitions define acceptable message between the two agents at a given time. Final states indicate the end of the conversation.

In InfoSleuth, we currently define conversation policies based on the type of the initial performative. For instance, Figure 1 shows a SUBSCRIBE conversation policy. Other policies we support include TELL, ASK-ONE, ASK-ALL, UPDATE, PERIODIC, and STANDBY (not all of which are “standard” KQML).

Sometimes the agent that receives the initial performative may decide to delegate the conversation to another agent. For example, an agent that plans and executes tasks in InfoSleuth may delegate the processing of certain kind of query to an agent that specializes in that type of query processing. This query processing agent would then take over the responding end of the ASK-* conversation, dealing directly with the agent that sent the query.

6.3 Implementation Approach

To implement prescriptive conversation policies in InfoSleuth, we have incorporated a *conversation layer* into our architecture. The conversation layer defines and enforces the different conversations available to the agents in the InfoSleuth system, and is accessed via a clearly defined API.

The conversation layer sits “on top of” the *generic agent* shell from which all our agents are being built. Thus, all InfoSleuth agents must communicate with other agents by using the conversation layer, which manages both incoming and outgoing conversations, and determines their legality.

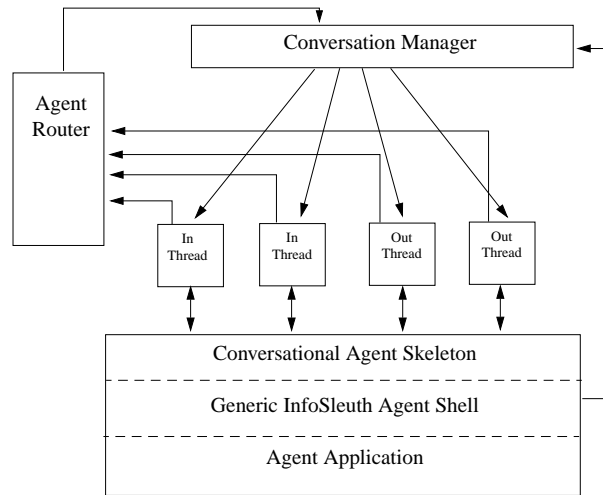


Fig. 2. Implementation of the InfoSleuth conversation layer.

However, a key feature of the conversation layer with respect to our development of an open multiagent system, is that it has been cleanly factored from the agent shell. This means that it may be used via its API by new agents which do *not* support the shell. In this way conversational policies are enforced without requiring any assumptions about the underlying agent implementation³.

The conversation layer is implemented by several components:

1. a *router*, which dispatches messages to and receives messages from the KQML implementation layer;
2. a *conversational agent skeleton*, which dispatches messages to and receives messages from the local agent, and
3. a *conversation manager*, which coordinates both incoming and outgoing conversations via “in” and “out” conversation threads.

Each conversation in the multiagent system’s conversation policy has an “out thread” class to encode its state machine from the perspective of the initiating agent and an “in thread” class to encode the state machine from the recipient’s perspective. Instances of these classes are instantiated as appropriate by the conversation manager. Each thread maintains state, performs error checking, and communicates with the agent for the duration of its conversation. Figure 2 shows the conversation layer architecture.

Thus, the first step in adding a new conversation to the conversation layer is to map its state machine — describing the pairwise conversation — to pairs of state machines which specify a conversation from the point of view of both the initiator and the recipient. Figure 3 shows an example of such a mapping, for simple ASK conversations. Once the incoming and outgoing state machines for a conversation have been specified, new

³ The conversation layer is written in Java, so does require a Java wrapper around any agents which utilize it.

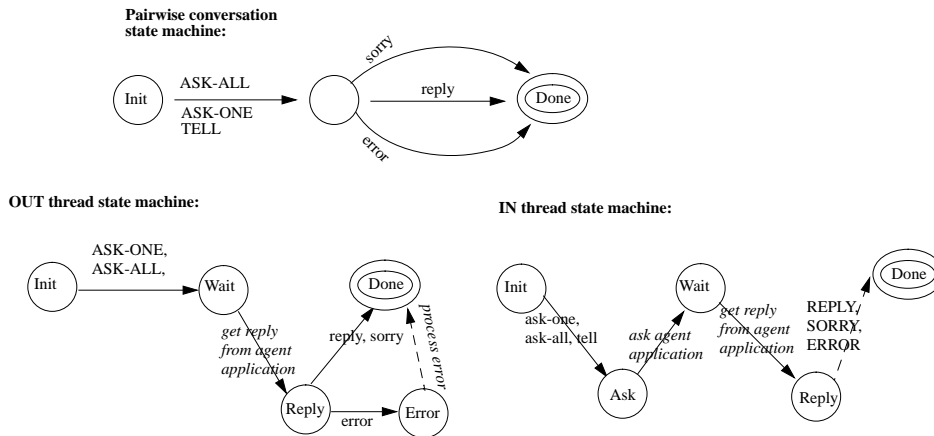


Fig. 3. State machines showing initiator and responder policies for a subset of the performatives.

classes are implemented to support the conversation, and are added to the conversation manager’s repertoire.

6.4 Conversation Policies and Core Agent Functionality

Section 6.3 described how the InfoSleuth conversation layer enforces the agents’ use of defined conversation policies: it ensures that the performatives output from an agent, and the incoming performatives that it gets, are correct with respect to the defined conversations of the policy. However, the conversation layer itself does not specify whether or not the functionality of the “core” agent matches the conversation layer. For example, the conversation layer ensures that the agent will never send anything but a REPLY (or SORRY or ERROR) in response to an ASK-* performative, but it does not ensure that the core agent “knows” that it needs to REPLY to an ASK. In many cases in our architecture, the correctness of the agents’ behavior with respect to the conversation policies remains implicit in the agents’ procedural code.

However, some of the InfoSleuth agents employ a rule-based core. These agents use a declarative knowledge base to specify the task plans, or partial ordering of actions, that are carried out in response to an incoming performative of a specific type. The actions, or *operators*, include what might be termed conversational operators. These operators create and send performatives, and process incoming messages. Static analysis of these agents’ knowledge bases, and comparison with a declarative specification of the conversation policy, can determine before run-time whether the agents’ actions are consistent with the policy⁴ome operators have procedural attachments. These can not be as easily analyzed, but the effects of each operator can be modularly tested and specified. When agents represent their functionality in terms of task plans, the implicit relationship between the conversation layer and the core agent can be made more explicit, increasing the expectation that new agents behave as desired when added to a system.

⁴ S

Some agent systems do not employ a conversation layer, but enforce a conversational policy via rule-based agents which all share the same conversation rules [1]. This approach works very well in a system of agents that all support rule-based reasoning. The InfoSleuth architecture allows agents to exploit rules for conversation when they have that ability, but does not require this ability in an open system — the conversation layer enforces conversational consistency even in purely procedural agents.

7 Conclusion

In this paper, we have proposed several keys necessary for openness in a multiagent system, and have discussed the implementation approach that InfoSleuth takes to address them. This approach includes the generation of a service ontology (and the need to push service activities from KQML down to the content of a message), the specification of common prescriptive conversation policies, and the creation of a conversation policy layer in the InfoSleuth architecture. These key issues have emerged from our past and current efforts to expand InfoSleuth by adding new agents with new functionality and with our ongoing efforts to interoperate with several other information-gathering agent systems.

We hint at three orthogonal sets of functionality that an agent communication layer deals with: routing services (including associating reply messages with their request messages), agent capability description, and lastly the content semantics which pertain to a specific task. We believe that there should be a clean factoring of these functions within the communication infrastructure. This would imply, for example, that routing services should be at a separate message layer from capability or task descriptions, and capability descriptions should be specified over a different ontology from task descriptions.

Future work in this area will include KQML interoperability experiments with other multiagent systems, investigation of the issues in using CORBA as a transport layer below KQML, and further investigation and tracking of the relationship between the KQML and FIPA efforts. Future research issues include understanding and increasing the scalability of the InfoSleuth architecture — more specifically, how performance and reliability are affected by the addition of numbers of agents of various types to the multiagent system; and automatic incorporation of new, declaratively-specified conversational policies into an agent system.

Another issue relates to open systems and the modeling of agent state. A conversation policy ensures that the agents in the open system all send the same sequences of performatives to each other. However, it says nothing about the agents' internal states and their interpretation of the semantics of the speech acts — e.g., whether or not they remember the contents of a TELL. We would like to explore the issues in open systems when agents model each other's internal states in this manner, and have expectations of other agent's states which are not true.

References

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

2. R. Bayardo, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. Infosleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of SIGMOD '97*, 1997.
3. J. Bradshaw, S. Dutfield, P. Benoit, and J. Woolley. KAoS: Toward an industrial-strength open agent architecture. In J.M. Bradshaw, editor, *Software Agents*, chapter 17. AAAI Press, 1997.
4. P. Cohen and H. Levesque. Communicative actions for artificial agents. In *ICMAS-95*, 1995.
5. K. Decker, K. Sycara, and M. Williamson. Modeling information agents: Advertisements, organizational roles, and dynamic behavior. In *Working Notes of the AAAI-96 workshop on "Agent Modeling"*, 1996.
6. T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J.M. Bradshaw, editor, *Software Agents*. AAAI Press, 1997.
7. T. Finin, C. Thirunavukkarasu, A. Potluri, D. McKay, and R McEntire. On agent domains, agent names and proxy agents. In *ACM CIKM Intelligent Information Agents Workshop*, 1995.
8. T. Finin and G. Wiederhold. An overview of KQML: A knowledge query and manipulation language. Technical report, available through the Stanford CS Dept., 1991.
9. D. Geddis, M. Genesereth, A. Keller, and N. Singh. Infomaster: A virtual information system. In *Intelligent Information Agents Workshop at CIKM '95*, 1995.
10. N. Jacobs and R. Shea, "The Role of Java in InfoSleuth: Agent-based Exploitation of Heterogeneous Information Resources", IntraNet96 Java Developers Conference, April, 1996.
11. Y. Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland at Baltimore County, 1996.
12. McGuire, Kuokka, Weber, Tenenbaum, Gruber, and Olsen. SHADE: Technology for knowledge-based collaborative engineering. *Journal of Concurrent Engineering: Research and Applications*, 1(3), 1993.
13. C. Petrie, "<http://cdr.stanford.edu/ProcessLink/kqml-proposed.html>"
14. I. Smith and P. Cohen. Toward a semantics for an agent communications language based on speech-acts. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 24–31. AAAI, AAAI Press/The MIT Press, 1996.
15. T. Winograd and F. Flores. *Understanding Computers and Cognition*. Addison-Wesley, 1998.
16. D. Woelk, M. Huhns and C. Tomlinson. "InfoSleuth Agents: The Next Generation of Active Objects", *Object Magazine*, July/August, 1995.
17. D. Woelk and C. Tomlinson, "The InfoSleuth Project: Intelligent Search Management via Semantic Agents", *Second International World Wide Web Conference*, October, 1994.
18. C. Zaniolo, "The Logical Data Language (LDL): An Integrated Approach to Logic and Databases", *MCC Technical Report STP-LD-328-91*, 1991.
19. "<http://www.cselt.stet.it/fipa>"