Agent Communication Languages: The Current Landscape

Yannis Labrou, Tim Finin, and Yun Peng, University of Maryland, Baltimore County

The most noble and profitable invention of all other was that of speech, consisting of names or appellation, and their connection; whereby men register their thoughts, recall them when they are past, and also declare them one to another for mutual utility and conversation; without which there had been amongst men neither commonwealth, nor society, nor contract, nor peace, no more than amongst lions, bears, and wolves.

—Thomas Hobbes
Leviathan, Part I, Chapter IV

DESPITE THE SUBSTANTIAL NUMBER OF MULTIAGENT SYSTEMS THAT USE AN AGENT COMMUNICATION LANGUAGE, THE DUST HAS NOT YET SETTLED OVER THE ACL LANDSCAPE. ALTHOUGH SEMANTIC SPECIFICATION ISSUES HAVE MONOPOLIZED THE DEBATE, OTHER IMPORTANT PRAGMATIC ISSUES MUST BE RESOLVED QUICKLY IF ACLS ARE TO SUPPORT THE DEVELOPMENT OF ROBUST AGENT SYSTEMS.

ALTHOUGH THE FIRST PRIMATES that could be considered our remote ancestors appeared one to two million years ago, it has been less than 50,000 years since our species "invented" what we would recognize today as a language. In the short evolutionary time since that innovation, we have managed to raise ourselves to a level unmatched by other creatures on our planet. Admittedly, the attribution of this dramatic advancement solely to language cannot be proven, but language must have something to do with it. Is it a mere coincidence that the emergence of language coincides with the beginning of complex, long-lived communities? Obviously, language fostered complex interactions between the members of a community. And beyond facilitating day-to-day business, language granted memory. It allowed communities to convey the experiences and acquired knowledge of its members to the next generation.

Not all languages are spoken "natural languages" like English, Greek, or Mandarin Chinese. The signs that baseball managers, coaches, and players use and sign language for the deaf are examples of languages that exhibit a fundamental property of useful languages: the meanings of their tokens are shared. This is not to say that one cannot have a private language, whose symbols are understood only by oneself; however, there is not much you can do with such a language. If this private language becomes public, well, now we're talking. This leads us to a fundamental characteristic of any language: languages exist to serve a purpose, namely the communication between willing—and occasionally unwilling—participants.

Whereas evolution was the engine of language development for human agents, standardization efforts have assumed that role for software agents. Agents[1,2]—by which we always mean *software agents* in this article—suggest a paradigm for software development that emphasizes autonomy both at design time and runtime, adaptivity, and cooperation. This approach seems appealing in a world of distributed, heterogeneous systems. Languages for communicating agents promise to play the role that natural languages played for their human counterparts. An agent communication language that allows agents to interact while hiding the details of their internal workings will result in agent communities that tackle problems no individual agent could.

In this article, we introduce some concepts useful in discussing agent communication languages and then compare and evaluate the two major ACLs. (Throughout the article, we use the abbreviation ACL to refer both to an agent communication language as a concept and to ACLs collectively. There is an ACL simply named ACL, but we hope that context will prevent confusion.) Knowledge Query and Manipulation Language is our vehicle for introducing the fundamental notions of an ACL. The semantics of KQML have been the single most important issue in the debate over ACLs, and we include a brief overview of the arguments. The second ACL we discuss is FIPA ACL. This is the language developed by the Foundation for Intelligent Physical Agents, the first organized effort focusing on developing standards in the broader area of agents. In our comparative evaluation of KQML and FIPA ACL, we look beyond the dominant issue of semantics and point to the practical limitations that these ACLs share. As we show with reference to a few of the systems that have used ACLs, these practical problems are the central areas of immediate concern and future work for the ACL standardization effort.

Basic concepts of ACL

An ACL provides agents with a means of exchanging information and knowledge; Michael R. Genesereth has gone as far to equate agency with the ability of a system to exchange knowledge using an ACL.[3] Of course, other means have been used to achieve the lofty goal of seamless exchange of information and knowledge between applications. From remote procedure call and remote method invocation (RPC and RMI) to CORBA and object request brokers, the goal has been the same. What distinguishes ACLs from such past efforts are the objects of discourse and their semantic complexity. ACLs stand a level above CORBA for two reasons:

- ACLs handle propositions, rules, and actions instead of simple objects with no semantics associated with them.
- An ACL message describes a desired state in a declarative language, rather than a procedure or method.

But ACLs by no means cover the entire spectrum of what applications might want to exchange. Agents can and should exchange more complex objects, such as shared plans and goals, or even shared experiences and long-term strategies.

At the technical level, when using an ACL, agents transport messages over the network using a lower-level protocol—for example, SMTP, TCP/IP, IIOP, or HTTP. The ACL itself defines the types of messages (and their meanings) that agents can exchange. Agents do not, however, just engage in single-message exchanges; they have *conversations*—task-oriented, shared sequences of messages that they follow, such as a negotiation or an auction. At the same time, a higher-level conceptualization of the agent's strategies and behaviors drives the agent's communicative (and noncommunicative) behavior.

Traditionally, we understand the message types of ACLs as *speech acts*, which in turn we usually describe and define in terms of beliefs, desires, intentions, and similar modalities. This kind of intentional-level description can be just a useful way to view a system, or it can have a concrete computational aspect. The second case describes a large range of BDI agents—*belief, desire*, and *intention* agents—that have some implicit or explicit representation of the corresponding modalities. This representation is built on top of a substrate that describes the conceptual model of the agent's knowledge, goals, and commitments, commonly known as a BDI theory.

BDI theories and agents have faced criticisms over the number and choice of modalities and over the fact that multimodal BDI logics do not have complete axiomatizations and are not efficiently computable. Nonetheless, they offer an appealing framework to account for agent communications: when communicating, agents communicate their BDI states or attempt to alter their interlocutors' BDI states.

The origin of ACLs

The Knowledge Sharing Effort,[4,5] initiated circa 1990 by the Defense Advanced Research Projects Agency of the US Department of Defense, enjoyed the participation of dozens of researchers from both academia and industry. Its goal was to develop techniques, methodologies, and software tools for knowledge sharing and reuse at design, implementation, or execution time. The central concept of the KSE was that knowledge sharing requires communication, which in turn requires a common language; the effort focused on defining that common language. In the KSE model, software systems are virtual knowledge bases that exchange propositions using a language that expresses various complex attitudes. The proper term is *propositional attitudes*. Propositional attitudes are three-part relationships between

- an agent,
- a content-bearing proposition (for example, *it is raining*), and
- a finite set of propositional attitudes an agent might have with respect to the proposition (for example, believing, asserting, fearing, wondering, hoping, and so on).

For example, $<a, fear, raining(t_{now})>$ is a propositional attitude.

Although agents were not originally part of the KSE vocabulary, the group's conceptual breakdown of the common-language problem is applicable to what we currently refer to as agents. Expressions in a given agent's native language should be understood by some other agent that uses a different implementation language and domain assumptions.

So, according to the KSE, the first layer of the common-language problem is that of syntactic translation between languages in the same family or between families of languages. The Object Management Group (OMG) standardization effort is an example of work in this direction within the object-oriented language family.

Another layer is concerned with guaranteeing that tokens' semantic content is preserved among applications. In other words, the same concept, object, or entity must have a uniform meaning across applications even if different applications use different names to refer to it. Every agent incorporates some view of the domain (and the domain knowledge) it applies to. The technical term for this body of background knowledge is *ontology*. More formally, an ontology is a particular conceptualization of a set of objects, concepts, and other entities about which knowledge is expressed, and of the relationships among them. An ontology consists of terms, their definitions, and axioms relating to them;[6] terms are normally organized in a taxonomy.

A final layer addresses the communication between agents. This is not about transporting bits and bytes between agents; agents should be able to communicate complex attitudes about their information and knowledge content. Agents need to question other agents, inform them, request their services for a task, find other agents who can assist them, monitor values and objects, and so on. In an open environment, a simple RPC cannot provide these functions. Agents issue requests by specifying not a procedure but a desired state in a declarative language—that is, in some ACL.

The KSE view was that these layers are

independent of one another. The ACL is only concerned with capturing propositional attitudes regardless of how propositions are expressed. Still, propositions are what agents will be "talking" about.

A proposal within the KSE was to use Knowledge Interchange Format (KIF),[7] a particular logic language, as a standard for describing things within computer systems such as expert systems, databases, intelligent agents, and so on. Moreover, KSE researchers designed KIF specifically to make it useful as an *interlingua*. By this, we mean a language that is useful as a mediator in the translation of other languages. KIF is a prefix version of first-order predicate-order calculus with extensions to support meta-operators and definitions. The language description includes both a specification for its syntax and one for its semantics.

Ontolingua,[8] a language designed for describing ontologies, and a variety of supporting tools became the KSE solution to the problem of developing and maintaining ontologies. Researchers at Stanford's Knowledge Systems Laboratory have developed a set of tools and services to support the process of geographically distributed groups achieving consensus on common shared ontologies. These tools, built around Ontolingua, use the World Wide Web to enable wide access and let users publish, browse, create, and edit ontologies stored on an ontology server. Users can quickly assemble a new ontology from a library of existing modules, extend the result with new definitions and constraints, check for logical consistency, and publish the result in the library.

KQML: concepts of ACLs

Existing ACLs are KQML, its many dialects and variants, and FIPA ACL. KQML[9–10] illustrates the basic concepts of all these. With the exception of ACL, a KQML variant that assumes KIF as the content language, all KQML dialects and FIPA ACL follow the basic concepts of KQML that we discuss here.

KQML is a high-level, message-oriented communication language and protocol for information exchange independent of content syntax and applicable ontology. Thus, KQML is independent of the transport mechanism (TCP/IP, SMTP, IIOP, or another), independent of the content language (KIF, SQL, STEP, Prolog, or another), and independent of the ontology assumed by the content.

```
(ask-one
     :sender joe
     :content (PRICE IBM ?price)
     :receiver stock-server
     :reply-with ibm-stock
     :language LPROLOG
     :ontology NYSE-TICKS)
```
(a)

```
(tell
     :sender stock-server
     :content (PRICE IBM 14)
     :receiver joe
     :in-reply-to ibm-stock
     :language LPROLOG
     :ontology NYSE-TICKS)
```
(b)

Figure 1. Examples of messages in KQML: (a) a query from agent joe about the price of IBM stock and (b) the stock-server's reply.

Three-layer organization. Conceptually, we can identify three layers in a KQML message: content, communication, and message. The content layer bears the actual content of the message in the program's own representation language. KQML can carry any representation language, including languages expressed as ASCII strings and those expressed using binary notation. Every KQML implementation ignores the content portion of the message, except to determine where it ends.

The communication layer encodes a set of features to the message that describe the lower-level communication parameters, such as the identity of the sender and recipient, and a unique identifier associated with the communication.

The message layer, which encodes a message that one application would like to transmit to another, is the core of KQML. This layer determines the kinds of interactions one can have with a KQML-speaking agent. The message layer's primary function is to identify the network protocol with which to deliver the message and to supply a speech act or *performative* that the sender attaches to the content. This speech act indicates whether the message is an assertion, a query, a command, or any other of a set of known performatives. (KQML has adopted the term *performative* to mean any of its primitive message types. In speech act theory, a performative is an utterance that succeeds simply because the speaker says or asserts it. In English, such utterances typically appear in a first-person, present-tense, declarative form, often accompanied by "hereby"—for example, "I hereby request you to turn on the computer." Philip R. Cohen has argued[11] that *performative* is a poor term to use for all ACL primitive message types, because not all can be construed as actions that the sender can make so just by sending them. For historical reasons, however, we continue to use the term for KQML.)

In addition, since the content is opaque to KQML, the message layer also includes optional features that describe the content language, the ontology it assumes, and some type of description of the content, such as a descriptor naming a topic within the ontology. These features make it possible for KQML implementations to analyze, route, and properly deliver messages whose content is inaccessible.

Syntax and performatives. The syntax of KQML is based on the familiar *s-expression* used in Lisp—that is, a balanced parenthesis list. The initial element of the list is the performative; the remaining elements are the performative's arguments as keyword/value pairs. Because the language is relatively simple, the actual syntax is not significant and can be changed if necessary in the future. The syntax reveals the roots of the initial implementations, which were done in Common Lisp; it has turned out to be quite flexible.

A KQML message from agent joe representing a query about the price of a share of IBM stock might be encoded as shown in Figure 1a. In this message, the KQML performative is ask-one, the content is `(PRICE IBM ?price)`, the ontology assumed by the query is identified by the token `NYSE-TICKS`, the receiver of the message is to be a server identified as `stock-server`, and the query is written in a language called `LPROLOG`. The value of the `:content keyword` is the content level; the values of the `:reply-with`, `:sender`, and `:receiver` keywords form the communication layer; and the performative name with the `:language` and `:ontology` keywords form the message layer. In due time, the stock-server might send joe the KQML message in Figure 1b.

Although KQML has a predefined set of reserved performatives, it is neither a minimal required set nor a closed one. A KQML agent might choose to handle only a few (perhaps one or two) performatives. The set is extensible; a community of agents might choose to use additional performatives if they agree on their interpretation and the protocol associated with each. However, an implementation that chooses to implement one of the reserved performatives must implement it in the standard way.

One of the design criteria for KQML was to produce a language that could support a wide

tell(A,B,X)	
Pre(A):	Bel(A,X) $\wedge$ Know(A,Want(B,Know(B,S)))
Pre(B):	Int(B,Know(B,S))
where *S* may be any of Bel(B,X), or $\neg$(Bel(B,X)).	
Post(A):	Know(A,Know(B,Bel(A,X)))
Post(B):	Know(B,Bel(A,X))
Completion:	Know(B,Bel(A,X))

Figure 2. KQML semantics for *tell.*

variety of interesting agent architectures. Thus, KQML introduces a small number of KQML performatives that agents use to describe the metadata specifying the information requirements and capabilities; it also introduces a special class of agents called *communication facilitators*.[3] A facilitator is an agent that performs various useful communication services, such as maintaining a registry of service names, forwarding messages to named services, routing messages based on content, matchmaking between information providers and clients, and providing mediation and translation services.

Semantics. During its first few years of use, KQML existed with only an informal and partial semantic description. Critics identified this as one of its shortcomings.[11] During the past few years, researchers have put forth several efforts to provide a formal semantics.

In other works,[12–14] the first two authors of this article provide the semantics of KQML in terms of *preconditions*, *postconditions,* and *completion conditions* for each performative. Assuming a sender A and a receiver B, preconditions indicate the necessary states for an agent to send a performative, **Pre(A)**, and for the receiver to accept it and successfully process it, **Pre(B)**. If the preconditions do not hold, the most likely response is *error* or *sorry*.

Postconditions describe the states of the sender after the successful utterance of a performative, and of the receiver after the receipt and processing of a message but before a counterutterance. Postconditions **Post(A)** and **Post(B)** hold unless a *sorry* or an *error* is sent as a response to report the unsuccessful processing of the message.

A completion condition for the performative, **Completion,** indicates the final state, after, for example, a conversation has taken place and the intention associated with the performative that started the conversation has been fulfilled.

Establishing the preconditions for a performative does not guarantee its successful execution and performance. The preconditions only indicate what can be assumed to be the state of the interlocutors involved in an exchange. Similarly, the postconditions describe the states of the interlocutors assuming the successful performance of the communication primitive. Preconditions, postconditions, and completion conditions describe states of agents in a language of mental attitudes (belief, knowledge, desire, and intention) and action descriptors (for sending and processing a message). No semantic models for the mental attitudes are provided, but the language used to describe agents' states severely restricts the ways the mental attitudes can be combined to compose agents' states.

Figure 2 shows an example of semantics for sender A and receiver B in this framework. This semantics for *tell* suggests that an agent cannot offer unsolicited information to another agent. We can easily amend this by introducing another performative—let's call it *proactive-tell*—that has the same semantic description as *tell* but with **Pre(A)** being Bel(A,X), and an empty **Pre(B)**.

Another semantic approach[11,15] builds on earlier work on defining rational agency.[16] This body of work deems the term *performative* inappropriate to describe KQML's communication primitives; the suggested approach views the language's reserved message types as *attempts* at communication. These attempts involve two or more rational agents that temporarily form teams to engage in communication. This approach strongly links the ACL semantics to the agent theory assumed for the agents involved in an ACL exchange.

Foundation for Intelligent Physical Agents

The Foundation for Intelligent Physical Agents is a nonprofit association whose purpose is to promote the success of emerging agent-based applications, services, and equipment. FIPA's goal is to make available specifications that maximize interoperability across agent-based systems. As this description suggests, FIPA is a standards organization in the area of software agents. The organization originally included the word *physical* in its name to cover agents of the robotic variety. Over time, however, the adjective's presence has come to serve as a reminder that physical—that is, human—agents and interaction with them are part of the association's scope.

FIPA operates through the open international collaboration of member organizations, which are companies and universities active in the field. European and Far Eastern technology companies have been among the earliest and most active participants, including Alcatel, British Telecom, France Telecom, Deutsche Telecom, Hitatchi, NEC, NHK, NTT, Nortel, Siemens, and Telia.

FIPA's operations center around annual rounds of specification deliverables. The current specification is FIPA97, available at the FIPA home page, *www.fipa.org*. FIPA assigns tasks to technical committees, each of which has primary responsibility for producing, maintaining, and updating the specifications applicable to its tasks. The technical committee most important within the scope of this article is the one charged with producing a specification for an ACL. In addition, the agent management committee covers agent services such as facilitation, registration, and agent platforms; the agent/software interaction committee covers integration of agents with legacy software applications. Together, these three committees create the backbone of the FIPA specifications.

FIPA ACL. FIPA's agent communication language, like KQML, is based on speech act theory: messages are actions or communicative acts, as they are intended to perform some action by virtue of being sent. The FIPA ACL specification consists of a set of message types and the description of their pragmatics—that is, the effects on the mental attitudes of the sender and receiver agents. The specification describes every communicative act with both a narrative form and a formal semantics based on modal logic. It also provides the normative description of a set of high-level interaction protocols, including requesting an action, contract net, and several kinds of auctions.

FIPA ACL is superficially similar to KQML. Its syntax is identical to KQML's except for different names for some reserved primitives. Thus, it maintains the KQML approach of separating the outer language from the inner language. The outer language defines the intended meaning of the message; the inner, or content, language denotes the expression to which the interlocutors' beliefs, desires, and intentions, as described by the meaning of the communication primitive, apply.

KQML has been criticized for using the term *performative* to refer to communication primitives. In FIPA ACL, the communication primitives are called *communicative acts*, or CAs for short. Despite the difference in naming, KQML performatives and FIPA ACL communicative acts are the same kind of entity. To avoid confusion, we will use the terms *performative*, (communication) *primitive*, and *communicative act* interchangeably.

The FIPA ACL specification document claims that FIPA ACL (like KQML) does not make any commitment to a particular content language. This claim holds true for most primitives. However, to understand and process some FIPA ACL primitives, receiving agents must have some understanding of Semantic Language, or SL. We will discuss this important point later.

Semantics. SL is the formal language used to define FIPA ACL's semantics. SL is a quantified, multimodal logic with modal operators for beliefs (B), desires (D), uncertain beliefs (U), and intentions (persistent goals, PG). SL can represent propositions, objects, and actions. We can trace SL's origins to the work of Philip Cohen and Hector Levesque,[16] but its current form is primarily based on the work of M.D. Sadek.[17] A detailed description of SL, including its own semantics, is outside the scope of this article and can be found in the FIPA ACL specification.

In FIPA ACL, the semantics of each communicative act is specified as sets of SL formulae that describe the act's *feasibility preconditions* and its *rational effect*. For a given CA a, the feasibility preconditions FP(a) describe the necessary conditions for the sender of the CA. That is, for an agent to properly perform the communicative act a by sending a particular message, the feasibility preconditions must hold for the sender. The agent is not obliged to perform a if FP(a) holds, but it can if it chooses. A communicative act's *rational effect* represents the effect that an agent can expect to occur as a result of performing the action; it also typically specifies conditions that should hold true of the recipient. The receiving agent is not required to ensure that the expected effect comes about and might indeed find it impossible. Thus, an agent can use its knowledge of the rational effect to plan what CA to perform, but it cannot assume that the rational effect will necessarily follow.

$<i, inform(j, \phi)>$
FP: $B_i(\phi) \wedge \neg B_i(B_if_j(\phi) \vee U_if_j(\phi))$
RE: $B_j(\phi)$

Figure 3. FIPA ACL semantics for the communicative act *inform.* Agent *i* informs agent *j* of content ϕ.

Conformance with the FIPA ACL means that when agent A sends CA x, the FP(x) for A must hold. The unguaranteed RE(x) is irrelevant to the conformance issue.

This introduction should be enough for a basic understanding of the example in Figure 3, which shows the specification of the communicative act *inform*, in which agent i informs agent j of content ϕ. The content of *inform* is a proposition, and its meaning is that the sender informs the receiver that a given proposition is true. According to this semantics, the sending agent

- holds that the proposition is true ($B_i(\phi)$);
- does not already believe that the receiver has any knowledge of the truth of the proposition ($\neg B_i(B_if_j(\phi) \vee U_if_j(\phi))$); and
- intends that the receiving agent should also come to believe that the proposition is true (rational effect $B_j(\phi)$);

Comparing the ACLs

KQML and FIPA ACL are almost identical with respect to their basic concepts and the principles they observe. The two languages differ primarily in the details of their semantic frameworks. In one sense, this difference is substantial: because of the different semantic frameworks it would be impossible to come up with exact mappings or transformations between KQML performatives and their completely equivalent FIPA primitives, or vice versa. On the other hand, the ineluctable differences might be of little importance to many agents' programmers, if their agents are not true BDI agents. We will elaborate on this argument in the later section on the future of ACLs.

Both languages assume a basic noncommitment to a reserved content language. However, in the FIPA ACL case, as we mentioned, an agent must have some limited understanding of SL to properly process a received message (as in the case of the *request* CA). The two languages have the same syntax. That is, a KQML message and a FIPA ACL message look syntactically identical—except, of course, in their different names for communication primitives. This is an important attribute of FIPA ACL. FIPA changed the language's original, Prolog-like syntax to match KQML's to facilitate the transition of KQML systems to FIPA ACL. A large part of making an agent system communication-ready is to provide code that will parse incoming messages, compose messages for transport, and channel them through the network using a lower-level network protocol. This infrastructure will be the same regardless of the choice of ACL.

These encouraging thoughts do not apply to the semantics of the two languages. Following the KQML semantics described elsewhere,[12] we can see that semantically the two languages differ at the level of what constitutes the semantic description: preconditions, postconditions, and completion conditions for KQML; feasibility preconditions and rational effect for FIPA ACL. They also differ at the level of the choice and definitions of the modalities they employ (the language used to describe agents' states). Although we can approximate the KQML primitives in FIPA's framework and vice versa, a complete and accurate translation is not, in general, possible. For example, to define a CA in FIPA ACL that approximates KQML's *tell*, we can replace ϕ in the definition of *inform* with $B_i\phi$ (see Figure 3).

Another difference between the two ACLs is in their treatment of the registration and facilitation primitives. These primitives cover a range of important pragmatic issues, such as registering, updating registration information, and finding other agents that can be of assistance in processing requests. In KQML, these tasks are associated with performatives that the language treats as first-class objects. FIPA ACL, intended to be a purer ACL, does not consider these tasks CAs in their own right. Instead, it treats them as requests for action and defines a range of reserved actions that cover the registration and life-cycle tasks. In this approach, the reserved actions do not have formally defined specifications or semantics and are defined in terms of natural-language descriptions. Moreover, FIPA ACL does not currently provide facilitation primitives.

Many ACL users have expressed their desire that FIPA ACL include the facilitation primitives that they are accustomed to from KQML—*broker*, *recommend*, and *recruit*. Such user requests serve as a sobering reminder that to be practical, an ACL requires a careful mix of the theoretical and the pragmatic.

The emergence of FIPA ACL might be an

additional headache for implementers who must decide for themselves which one of the two ACLs to use. Our comments on the subject are bound to cause more headaches. Any system that is to use KQML (or FIPA ACL, for that matter) must provide the following things:

1. a suite of APIs that facilitate the composition, sending, and receiving of ACL messages;
2. an infrastructure of services that assist agents with naming, registration, and basic facilitation services (finding other agents that can do things for your agent); and
3. code for every reserved message type (performative or communicative act) that takes the action(s) prescribed by the semantics for the particular application; this code depends on the application language, the domain, and the details of the agent system using the ACL.

Ideally, a programmer should only have to provide item 3. Items 1 and 2 should be reusable components that the programmer integrates into the application code. Actually, the programmer should not even have to integrate the things listed under item 2; they ought to exist as a continuous running service available to any new agent. But the sad truth is that these services have not been the focus of ACL standardization efforts. No service exists through which one can register an agent by just sending a registration message. The disagreement over such services has resulted, in part, in a multitude of APIs. (The other reasons for the multitude of APIs are the minor syntactic differences between the various varieties of KQML and the different naming schemes employed by the various KQML-speaking agent systems.)

Every multiagent system that uses an ACL has a homegrown implementation of these APIs—there are more than a handful of APIs written in Java, for Java agents—and its own infrastructure of basic services. Providing the code that processes the primitives is more of an art than a science. Existing semantic approaches rely on multimodal logics that are often noncomputable or have no efficient implementation. The process of grounding the theory into code would result in a system that differs substantially—and probably unpredictably—from the theory on paper. To make matters worse, if the code does not implement the modalities assumed by the semantics, the programmer will most likely follow his or her intuitive understanding of the semantics of the communication primitives.

In theory, the similarity in basic assumptions and syntax among existing ACLs means that only item 3 from our list of requirements should change according to the choice of ACL. Even then, much to the dismay of those defining ACL semantics, the implementers' intuitive understanding of the primitives might prevail over the concise semantic definitions. So, unless an agent implements modalities (such as belief, desire, intention, and so on) following the particular agent theory that the semantic account suggests, the decision of which language to use should be based on pragmatic concerns.

ACL-supporting systems and applications

Over the past few years, we have seen the emergence of a multitude of applications and systems built around ACLs. Instead of providing a compendium of such systems, we focus here on a few whose features and characteristics exemplify current approaches and trends. Each of the systems we discuss falls into one of two categories:

- applications, that is, multiagent systems that use an ACL for interagent communication, or
- APIs that facilitate the incorporation of ACL-speaking capabilities into an application.

Since the ACL itself is an abstraction—a collection of communication primitives deemed useful for higher-level communication between agents—there is no such thing as an "implementation" of an ACL.

All the systems we mention here use some variant of KQML as their ACL. As of the spring of 1998, there were no published, deployed systems claiming to use FIPA ACL.

Infosleuth[18,19] is a project by MCC that emphasizes the semantic integration of heterogeneous information in an open dynamic environment. The communicating agents, primarily written in Java, make use of an infrastructure of basic services (agents) for authentication, brokering, monitoring, and visualization of the agents' interaction. An integral part of the architecture is the ontology agent, which assists with the semantic integration of the information handled. Infosleuth agents engage in conversations rather than single-message exchanges.

Knowledgeable Agent-Oriented System[20] is a Boeing project aimed at providing an infrastructure for agent development. Kaos relies heavily on object-oriented technology; it uses, for example, a CORBA-based message delivery mechanism. It also emphasizes persistent interaction between agents that take into account not only the particular communication primitive but the content of the message and the applicable conversation policies. The system allows the design of agents that support specialized suites of interactions.

Infomaster[21] is an information integration system from Stanford that uses ACL, the KQML variant with KIF as its content language. The resulting language does not observe the distinction between the content layer and the message layer. Infomaster integrates structured information sources, giving the illusion of a centralized, homogeneous information system.

Java Agent Template, Lite—or JATLite—is a package of Java programs, developed at Stanford, that allow users to create communicating agents quickly. Agents run as applets launched from a browser, and for that reason all agents register with an agent message router facilitator that handles message delivery.

The Java-based Agent Framework for Multi-Agent Systems[22,23] is a set of classes that support implementing communicating agents in Java. Developed at the University of Cincinnati, JAFMAS supports directed (point-to-point) communication as well as subject-based, broadcast communications. JAFMAS, which includes support for conversations, provides the environment for AARIA,[24] a planning and scheduling project for manufacturing.

Jackal,[25] developed at University of Maryland, Baltimore County, is another Java package that allows applications written in Java to communicate via an ACL. KQML is currently the ACL of choice for this package, but it could easily support FIPA ACL. Jackal is currently in use in the CIIMPLEX project,[26,27] another project that involves planning and scheduling for manufacturing. Jackal strongly emphasizes conversations between agents. It provides a flexible framework for designing agents around conversations and includes extensive support for registration, naming, and control of agents.

A consideration of these projects leads us to the following points:

- Java is rapidly becoming the language of choice. Implementing BDI agents with traditional AI languages is problematic enough, but we have little experience and fewer tools for doing so with object-oriented languages like Java. This raises again the problems of existing semantic approaches and conformance to them.

- Many of the new APIs support conversations, an intuitive way of structuring an agent's activities. Given the problematic nature of compliance with the ACL's semantic account, conversations shift the focus from the agent's internal workings to its observable behavior—the sequences of messages it sends to other agents. Agents can agree on a conversation protocol for a particular task—a negotiation or an auction, for example—and then engage in a scripted interaction. We do not suggest that this is a conformance test, but it might be useful for an agent designer to know that its interlocutors will engage in a scripted, prespecified communicative behavior.
- Each implementation introduces its own variety of supporting agents and services for tasks such as naming, authentication, monitoring, and brokering. We need some agreement on the assumptions of these services so that they can be provided as a standard suite of tools.

The future of ACLs

We do not believe that KQML and FIPA ACL are in conflict. They both express the same basic ideas about what an agent communication language is. KQML does not have an official body behind it orchestrating its evolution. At the same time, the KQML development model, based on experimentation and continuous feedback from its community of users, has helped KQML grow with an emphasis on the practical concerns of agent development. FIPA ACL does not have a community of users because no FIPA ACL applications have appeared yet, and thus it is untested in practice. It does, however, enjoy the support of an organization with a concrete, comprehensive agenda. In the immediate future, FIPA ACL's choices will be put to the test as applications that use it are deployed. A new DARPA-sponsored initiative in the area of agents promises to help guide the next iteration of ACLs in the US research community.

Semantics have dominated the debate surrounding ACLs. Despite the substantial amount of work on this problem, the issue of an agent's conformance with the ACL semantics (assuming approaches such as those outlined in this article) remains thorny.[28] This issue puts into question the degree of usefulness of such semantic accounts. In the near future, more pragmatic concerns ought to be addressed. Offering naming and registration services along with basic brokering facilities should be among the immediate goals of the ACL community.

Another area that requires attention is that of defining basic ontologies for speaking about agents and their query-answering capabilities and requirements. Existing ACLs offer a rather narrow and inflexible way for performing such tasks. Finally, it would be useful to standardize some of the basic conversation protocols for the more fundamental tasks. This would be of particular interest to programmers who have no affinity for the standard BDI approach.

All of these pragmatic issues are very important for the deployment of agent applications. The availability of these services and standards would reduce the overhead of agent development and allow us to shift our focus to what agents actually *do*. The design and development of the infrastructure for communication has consumed the time and resources of those involved with agent development, at the expense of compelling new applications that naturally fit the agent software paradigm. Within the FIPA community, there is an effort to address some of the infrastructure issues, but the process is still in an early stage. It remains to be seen to what extent that effort will incorporate the experiences and lessons of others in dealing with these issues.

We usually hear agents mentioned within the context of the Web, and the Internet is the arena in which we generally expect them to compete. But KQML and FIPA ACL have followed a path away from the mainstream Internet technologies and standards. No major player has manifested an interest in ACLs, and no Internet standardization organization has ACLs in its agenda. Should existing ACLs become more Internet friendly? And if so, how? Taking advantage of eXtended Markup Language (XML) and possibly Resource Definition Format (RDF) seems like a reasonable course of action, especially when it comes to describing agents' features and capabilities. But even at the syntactic level, how about replacing KQML's Lisp-like syntax with XML?

Our point is that an ACL is an abstract idea that over time has evolved to describe some concrete and relatively well-understood concepts. But this journey has taken place on the sidelines of the revolution we have been experiencing in the past few years. Continuously running services for agents and better integration with existing and emerging Web technologies might push ACLs into the field.

THE CONCEPT OF A STANDARD communication language for software agents that is based on speech acts has found wide appeal, both among researchers interested in working out the theory of agent communication and among those with the aim of engineering practical software systems. Many researchers believe that the development of an effective, rich ACL is one of the keys to the agent paradigm.

KQML was among the first such ACLs to be developed and used. Moreover, it is the only one to date that has enjoyed substantial use by people other than its developers. However, after eight years of experimentation and experience, there are still serious signs of immaturity: In general, different KQML implementations cannot interoperate. There is no fixed specification sanctioned by a consensus-creating body. Finally, there is no agreed-upon semantics foundation. Is the KQML experiment a failure?

We think not. KQML has played an important role in defining what an ACL is and what the issues are when it comes to integrating communication into agent systems. Although existing KQML implementations tend not to interoperate, this is mainly due to a lack of a real motivation. Agent-based systems research is still at an early stage, and there has been no benefit to individual research groups in focusing on interoperability issues. Although the lack of a sanctioned specification has probably impeded the adoption of KQML for many big projects, it has allowed much experimentation with dialects and variations on the theme. We hope that the current FIPA effort will supply the needed sanctioning body for the next iteration of a KQML-like language.

Finally, the semantics issue is in practice much less important than it sounds, especially since the problem of defining and identifying conformance to the semantics is not resolved. Given the possibility that it might be impossible to find a satisfactory solution to the semantic conformance problem, we computer scientists will be left with a justifiable sense of disappointment over a language that lacks formal, verifiable semantics. However, this disappointment need not touch the programmers who want to register their agents, find other agents, and send and receive ACL messages. What they will find much more disappointing is a lack of standard conventions for the basic agent services, such as naming, authentication, registration, capability definition, and facilitation. Our focus on semantic clarity and purity is partly responsible for the slighting of these crucial issues. After all, what good does it do to have an agent that conforms with some ACL semantic account if you cannot register your agent and send and receive ACL messages? ■

References

1. J. Bradshaw, ed., *Software Agents*, AAAI Press, Cambridge, Mass., 1997.
2. M. Huhns and M. Singh, eds. *Readings in Agents*, Morgan Kaufmann Publishers, San Francisco, 1997.
3. M.R. Genesereth and S.P. Katchpel, "Software Agents," *Comm. ACM*, Vol. 37, No. 7, 1994, pp. 48–53, 147.
4. R. Neches et al., "Enabling Technology for Knowledge Sharing," *AI Magazine*, Vol. 12, No. 3, Fall 1991, pp. 36–56.
5. R.S. Patil et al. "The DARPA Knowledge Sharing Effort: Progress Report," *Readings in Agents*, M. Huhns and M. Singh, eds., Morgan Kaufmann, 1997.
6. T.R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, Vol. 2, 1993, pp. 199–220.
7. M. Genesereth et al., *Knowledge Interchange Format, Version 3.0 Reference Manual*, tech. report, Computer Science Department, Stanford Univ., Stanford, Calif., 1992.
8. A. Farquhar, R. Fikes, and J. Rice, *The Ontolingua Server: A Tool for Collaborative Ontology Construction*, Tech. Report KSL-96-26, Stanford Knowledge Systems Laboratory, Stanford, Calif., 1996.
9. *Specification of the KQML Agent Communication Language*, tech. report, DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1993.
10. Y. Labrou and T. Finin, *A Proposal for a New KQML Specification*, Tech. Report TR-CS-97-03, Computer Science and Electrical Engineering Dept., Univ. of Maryland, Baltimore County, Baltimore, Md., 1997.
11. P.R. Cohen and H.J. Levesque, "Communicative Actions for Artificial Agents," *Proc. First Int'l Conf. Multi-Agent Systems (ICMAS'95)*, AAAI Press, 1995.
12. Y. Labrou, *Semantics for an Agent Communication Language*, doctoral dissertation, Computer Science and Electrical Engineering Dept., Univ. of Maryland, Baltimore County, 1996.
13. Y. Labrou and T. Finin, "Semantics and Conversations for an Agent Communication Language," *Readings in Agents*, M. Huhns and M. Singh, eds., Morgan Kaufmann, 1997.
14. Y. Labrou and T. Finin, "Semantics for an Agent Communication Language," *Agent Theories, Architectures and Languages IV*, M. Wooldridge, J.P. Muller, and M. Tambe, eds., Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 1998.
15. I.A. Smith and P.R. Cohen, "Toward a Semantics for an Agent Communications Language Based on Speech-Acts," *Proc. 13th Nat'l Conf. Artificial Intelligence*, AAAI Press, 1996.
16. P.R. Cohen and H.J. Levesque, "Persistence, Intention, and Commitment," *Intentions in Communication*, P.R. Cohen, J. Morgan, and M.E. Pollack, eds., MIT Press, Cambridge, Mass., 1990, pp. 33–69.
17. M.D. Sadek, "A Study in the Logic of Intention," *Proc. Third Conf. Principles of Knowledge Representation and Reasoning (KR'92)*, Morgan Kaufmann, 1992, pp. 462–473.
18. R.J. Bayardo et al., "Infosleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Systems," *Proc. ACM Sigmod Int'l Conf. Management of Data*, ACM Press, New York, 1997.
19. M. Nodine and A. Unruh, "Facilitating Open Communication in Agent Systems: The Infosleuth Infrastructure," *Proc. Fourth Int'l Workshop on Agent Theories, Architectures, and Languages*, M. Singh, A. Rao, and M. Woolridge, eds., Springer-Verlag, 1997.
20. J.M. Bradshaw et al., "Kaos: Toward an Industrial-Strength Open Agent Architecture," *Software Agents*, J.M. Bradshaw, ed., AAAI Press, 1997.
21. M.R. Genesereth, A.M. Keller, and O.M. Duschka, "Infomaster: An Information Integration System," *Proc. ACM Sigmod Int'l Conf. Management of Data*, ACM Press, 1997.
22. D. Chauhan, *JAFMAS: A Java-Based Agent Framework for Multiagent Systems Development and Implementation*, master's thesis., Electrical Engineering and Computer Science Dept., Univ. of Cincinnati, Cincinnati, 1997.
23. D. Chauhan and A. Baker, "JAFMAS: A Multiagent Application Development System," *Proc. Second ACM Conf. Autonomous Agents*, M. Wooldridge and T. Finin, eds., ACM Press, 1998.
24. H.V.D. Parunak, A.D. Baker, and S.J. Clark, "The AARIA Agent Architecture: An Example of Requirements-Driven Agent-Based System Design," *Proc. First Int'l Conf. Autonomous Agents (ICAA'97)*, 1997, ACM Press, New York, pp. 482–483.
25. R.S. Cost et al., "Jackal: A Java-Based Tool for Agent Development," *Working Papers of the AAAI-98 Workshop on Software Tools for Developing Agents*, AAAI Press, 1998.
26. Y. Peng et al., "A Multi-Agent System for Enterprise Integration," *J. Applied Artificial Intelligence*, Vol. 1, No. 1, 1998.
27. B. Chu et al., "Toward Intelligent Integrated Manufacturing Planning-Execution," *The Int'l J. Agile Manufacturing*, Vol. 1, No. 1, 1998.
28. M. Wooldridge, "Verifiable Semantics for Agent Communication Languages," *Int'l Conf. Multi-Agent Systems (ICMAS'98)*, IEEE Computer Society Press, Los Alamitos, Calif., 1998.

Yannis Labrou is a research assistant professor of computer science and electrical engineering at the University of Maryland, Baltimore County. He is a founding member of the FIPA academy and has been an active participant in FIPA specification development. Prior to joining UMBC, Labrou worked as an intern in the Intelligent Network Technology group of IBM's T.J. Watson Research Center. He holds a BSc degree in physics from the University of Athens, Greece, and he received his PhD in computer science from UMBC in 1996. Contact him at 1000 Hilltop Circle, ECS 210, Univ. of Maryland, Baltimore County, Baltimore, MD 21250; jklabrou@cs.umbc.edu.

Tim Finin is a professor of computer science and electrical engineering at UMBC. For more than 25 years, his research has focused on the applications of artificial intelligence to problems in database and knowledge base systems, natural-language processing, intelligent interfaces, and robotics. He is currently working on the theory and applications of intelligent software agents. Finin holds an SB degree in electrical engineering from MIT and a PhD in computer science from the University of Illinois, Urbana-Champaign. Contact him at finin@cs.umbc.edu.

Yun Peng is an associate professor of computer science and electrical engineering at UMBC. His research interests include a variety of subjects in artificial intelligence, neural networks, and artificial life. He is currently working on intelligent software agents and their real-world applications. He received his BS degree in electrical engineering from Harbin Engineering Institute, China, his MS in computer science from Wayne State University and his PhD in computer science from the University of Maryland, College Park. Contact him at ypeng@cs.umbc.edu.