

Standardizing Agent Communication

Yannis Labrou

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, Maryland 21250
`jklabrou@csee.umbc.edu`

Abstract. An Agent Communication Language (ACL) is a collection of speech-act-like message types, with agreed-upon semantics, which facilitate the knowledge and information exchange between software agents. From Knowledge Query and Manipulation Language (KQML) to FIPA ACL, ACL's have been a cornerstone for the development of systems of communicating agents, and simultaneously they have been the subject of intensive standardization efforts. Standardization's goal is usability. As a result, although the initial focus on ACL's revolved around establishing the semantics of ACL's, a variety of usability-related questions have entered the picture of standardizing communication among agents. In this article, we present these questions and the work that addresses them, alongside the historical evolution of ACL's, their semantics and the results of their standardization.

1 Introduction

In the past ten years we have experienced a transformation of the term agents from a Artificial Intelligence term to a buzzword, often accompanied by promises of "amazing" feats that agents will be capable of. The term agents by which we always mean software agents now refers to a paradigm for software development, rather than a class of artifacts, which emphasizes autonomy both at design time and runtime, adaptivity, and cooperation; the agent paradigm seems appealing in a world of networked, distributed and heterogeneous systems. This transformation has inevitably affected the tools and methodologies used for software agent development, even though none of the promises has yet been delivered anywhere near fully.

A persistent theme throughout agents' conceptual evolution has been their ability to interact (communicate) with one another and thus be able to tackle collectively problems that no single agent can, individually. For a large part of the agents' community, the role of endowing agents with the ability to communicate was left to the Agent Communication Language (ACL). Knowledge Query and Manipulation Language (KQML), conceived in the early 90's gradually defined the concept of an ACL. KQML sprang out of the work of the Knowledge Sharing Effort (KSE), a consortium led by (mostly) AI researchers, which was aimed at achieving interoperability between knowledge bases. Initially, agents were not

part of the vocabulary of the KSE and KQML was thought of as an integral part of a larger solution to the interoperability question (see Section 2); KQML, as its name probably suggests, had nothing to do with agents. Early KQML can be summarized as a collection of speech-act-like message types, expressed as ASCII strings, with a LISP-like syntax, that are transported over TCP/IP connections, and aimed at knowledge and information exchange between software systems that are viewed as Virtual Knowledge Bases. KQML specifications and discussions about KQML at that time, manifested the AI-influenced thinking behind its design and the considerations that were deemed important. It is only 3-4 years after KQML's inception that it was recasted [18, 23] as a language for agent communication.

As a larger number of researchers (and users) became part of the fledging KQML community, KQML was dissociated from the waning KSE (and its concerns) and new issues came at the forefront. Many different groups designed and built multi-agent systems that used KQML for inter-agent communication and a large part of their efforts revolved around building the necessary infrastructure for sending and receiving properly formed KQML messages over the network, developing schemes for naming agents and mechanisms for distributing and sharing agent names and (perhaps most importantly) doing all that with traditional (non-AI) programming environments. Initially in C, and soon thereafter in Java, which was just emerging in the mid-90's, these implementations acted as experiments for resolving all these "little" things that the KQML specification had not addressed. As more time was spent dealing with these (seemingly "lower-level") issues, researchers realized that the list of issues was an ever expanding one. During the same period, the term "agents" came into usage to refer to a software-design paradigm rather than AI artifacts. As a result, integrating (and "implementing") KQML with primarily object-oriented code became another major issue.

These changes resulted to a shift of focus from what content (knowledge) the agent (Virtual Knowledge Base, in the KSE vocabulary) represents and which attitudes about that content are to be conveyed (old view, which inevitably emphasizes proper semantics of the communication primitives) to the act of communication itself and considerations about transport, concurrency, networking and architectural assumptions. In the older, earlier framework, the emphasis was on the semantics of the ACL's message types because the semantics properly define the attitudes suggested by the message types; an ACL is exactly that, i.e., a language of attitudes about content. This trend is manifested by the majority of the earlier work on ACLs [9, 23]. The later framework, though, emphasizes software development and focuses on how these attitudes are to be communicated among pieces of software (agents) that populate the network; the new focus is syntax and encoding, pragmatic considerations and software integration.

In discussing the standardization of Agent Communication Languages, we first trace the historic origins of KQML; we believe that situating KQML in the context of the Knowledge Sharing Effort will help the reader to better understand the evolution of ACL ideas and concepts. We briefly present KQML

and introduce the concepts necessary for discussing agent communication languages¹. The semantics of KQML have been the single most important issue in the early debates 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 argue that the developer is indifferent to the subtle semantic-layer differences. The ACL concept, as an agent development tool has evolved in a way that transcends the semantics and the concept now includes new issues. After presenting the new problems challenging ACL research, we examine the current work that addresses them.

Our thesis is that standardization of Agent Communication Languages, has, in fact, come to refer to standardizing Agent Communication, because other topics, some of which we explore here, are an integral part of designing and building challenging systems of communicating agents.

2 Origins of Agent Communication Language Concepts

Understanding the evolution of the Agent Communication Language concept requires understanding the context that gave birth to KQML. KQML was first introduced as one of the results of the Knowledge Sharing Effort (KSE²) [31] [34], which has influenced current efforts in inter-agent communication approaches.

The KSE was initiated as a research effort circa 1990 with encouragement and relatively modest funding from U.S. government agencies (DARPA especially). The KSE was highly active for roughly five years thereafter, and enjoyed the participation of dozens of researchers from both academia and industry; the researchers represented various branches of the AI community. Its goal was to develop techniques, methodologies and software tools for knowledge sharing and knowledge reuse between knowledge-based (software) systems, at design, implementation, or execution time. Agents, especially intelligent agents, are an important kind of such knowledge-based systems (other kinds include expert systems or databases, for example). The central concept of the KSE was that knowledge sharing requires communication, which in turn, requires a common language; the KSE focused on defining that common language

In the KSE model, agents (or, more generally, knowledge-based systems) are viewed as (virtual) knowledge bases that exchange propositions using a language that expresses various propositional attitudes. Propositional attitudes are three-part relationships between

- an agent,
- a content-bearing proposition (for example, *it is raining*), and

¹ 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.

² <http://www.cs.umbc.edu/kse/>

- 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, $\langle a, \textit{fear}, \textit{raining}_{\textit{now}} \rangle$ is a propositional attitude.

The KSE model includes three layers of representation: (1) specifying propositional attitudes; (2) specifying propositions (i.e., *knowledge*) - this is often called the (propositional) content layer; and (3) specifying the ontology [20](i.e., vocabulary) of those propositions. The KSE accordingly includes a component (with associated language) for each of these: Knowledge Query and Manipulation Language (KQML) for propositional attitudes, Knowledge Interchange Format (KIF ³) [17] for propositions, and Ontolingua [14].

Within the KSE approach, the three representational layers are viewed as mainly independent of another. In particular, the language for propositional content (i.e., the content language) can be chosen independently from the language for propositional attitudes. In other words, in the KSE approach, the role of an ACL, namely KQML's in the case of the KSE (or FIPA ACL's, much later) is only to capture propositional attitudes, regardless of how propositions are expressed, even though propositions are what agents are "talking" about ⁴.

KQML was influenced by the Virtual knowledge Base concept which emphasized propositional content and focused on defining the propositional attitudes. KQML was not concerned with all of the "mechanics" of the interaction/communication nor prescribed much about how an agent is designed and how communication is incorporated in this design. These were intentional choices of the original KQML specification. The intent was to allow for various specific design choices on these issues. Even the chosen syntax was deemed as changeable. But as we will show, building functioning KQML-speaking agents required agreement on certain choices, often on issues seemingly as obscure, as what is, for example, the terminating character of the ASCII stream that is a KQML message transmitted over a TCP connection.

3 KQML: Concepts of ACL's

Existing ACLs are KQML [1] [24], its many dialects and variants, and FIPA ACL. KQML 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,

³ <http://logic.stanford.edu/kif/> and <http://www.cs.umbc.edu/kif/>

⁴ In a similar spirit, the approach of the technical committee that worked on FIPA ACL is that the content language should be viewed as orthogonal to the rest of the ACL message type.

STEP, Prolog, or another), and independent of the ontology assumed by the content.

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 identity⁵ of the sender and recipient, and a unique identifier associated with the communication.
- The message layer, which encodes a message that an 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 function is to identify the speech act [2] 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 performatives⁶. 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.

3.1 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. The syntax reveals the roots of the initial implementations, which were done in Common Lisp; it has turned out to be quite flexible.

⁵ The *identity* is some symbolic name, e.g., *labrou*. One of the first identified problems was that these symbolic names provide no information about a program’s actual network address and they can only be useful in the context of some existing name space, to which there is no reference in the name itself.

⁶ KQML designers adopted the term performative to mean any of KQML 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*. Cohen has argued [38] 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.

A KQML message from agent "grososf" representing a query about the type of processor of a particular laptop may be encoded as shown in Figure 1a. In this message, the KQML performative is *ask-one*, the content is (*CPU libretto50 ?processor*), the ontology assumed by the query is identified by the token *laptop*, the receiver of the message is to be an agent identified as *laptop-center*, and the query is written in a language called *KIF*. The value of the `:content` keyword is the content layer; 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, *laptop-center* might send *labrou* the KQML message in Figure 1b.

```
(ask-one
  :sender labrou
  :receiver laptop-center
  :content (CPU libretto50 ?processor)
  :ontology electronics
  :language kif)
```

(a)

```
(tell
  :sender laptop-center
  :receiver labrou
  :content (CPU libretto50 pentium)
  :ontology electronics
  :language kif)
```

(b)

Fig. 1. Examples of messages in KQML: (a) a query from agent "grososf" about the type of CPU of a laptop and (b) a possible response.

Although KQML has a predefined set of reserved performatives, it is neither a minimal required set nor a closed one. A KQML agent may choose to handle only a few (perhaps one or two) performatives. The original KQML specification considered the set to be extensible; a community of agents might choose to use additional performatives if they agree on their interpretation. However, an implementation that chooses to implement one of the reserved performatives must implement it in the agreed-upon way.

One of the design criteria for KQML was to produce a language that could support a wide variety of interesting agent architectures. Thus, KQML introduces a small number of performatives that agents use to describe capabilities; it also introduces a special class of agents called communication facilitators, which are (ordinary) KQML-speaking agents that are capable of processing the aforementioned set of performatives. 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 con-

tent, matchmaking between information providers and clients, and providing mediation and translation services.

3.2 Semantics

During its first few years of use, KQML existed with only an informal semantic description. Critics identified this as one of its shortcomings [9]. During the past few years, researchers have put forth several efforts to provide a formal semantics.

In other works [23] [22] [27], Labrou and Finin 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, $\mathbf{Pre(A)}$, and for the receiver to accept it and successfully process it, $\mathbf{Pre(B)}$. If the preconditions do not hold, the most likely response will be one of the performatives *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 counter-utterance. Postconditions $\mathbf{Post(A)}$ and $\mathbf{Post(B)}$ 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 have been the state of the interlocutors involved in an exchange, just before it occurred (assuming conforming interlocutor agents). 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 (BEL, WANT, KNOW, INT) 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 $\mathbf{Pre(A)}$ being BEL(A,X), and an empty $\mathbf{Pre(B)}$.

Another semantic approach [9] [38] builds on earlier work on defining rational agency [8]. 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 co-operative communication. This approach strongly links the ACL semantics to the agent theory assumed for the agents involved in an ACL exchange.

tell(A,B,X) A states to B that A believes the content to be true.

- BEL(A,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))

Fig. 2. KQML semantics for *tell*

What the KQML specification did not specify

The first KQML specification introduced basic concepts about the language, some of the design assumptions, the syntax for the performatives, natural language descriptions of the meaning of the performatives and some examples. The community of KQML's users was expected to provide for all the other necessary elements for an implementationally viable language. It took the KQML community years to establish which were all the necessary elements.

The semantics were not part of the original KQML specification. Also, the specification did not offer much in terms of valid sequences of messages during agent interaction. Although the specification sporadically eluded to possible response performatives following a particular performative, such commentary was mostly intuitive suggestions. [23] was a first attempt to address these two issues, including an effort to specify *legal* sequences of performatives during agent interaction, using the term *conversation policies* to refer to such sequences. Conversation Policies (or conversation protocols, or just conversations) eventually became a separate thread of research in the ACL community for reasons we discuss later.

Some other issues were more practical in nature but equally important for agent development. The KQML specification assumed the existence of a name space, thanks to which, agents could be referenced by symbolic names; these symbolic names were presumably all that was needed for messages to reach their destination agents. No details of the name space were provided, nor was it explained how the associations between symbolic names and network addresses became known to new agents. Furthermore, just knowing a network address is not sufficient for initiating a KQML interaction, since an agent needs to know at least the network protocol (TCP, SMTP) that the receiving agent can process. Different groups made different choices on these (and many other) issues, since the KQML specification offered no prescribed way for addressing them.

In summary, the early KQML specification [1] described a concept (the Agent Communication Language) and a framework for future work; by no means, was it a specification for a working prototype or an implementation. Agent designers were implicitly asked to make their choices on a number of issues as they saw fit, as long as they stayed within the realm of the conceptual framework.

The exchange of ideas through research meetings and mailing lists established a shared understanding of the issues, even though particular solutions never made it into an official specification. A revised proposed specification made available in 1997 [24], was an attempt to provide a new starting point for discussions and to reconcile many of the then current ideas with the need for a more clear and consistent specification. At around the same time, the advent of the Foundation for Intelligent Physical Agents (FIPA) brought new life to the standardization question.

4 The Foundation for Intelligent Physical Agents (FIPA)

The Foundation for Intelligent Physical Agents is a nonprofit association whose purpose is to promote the success of emerging agent-based applications and services. 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 include 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, Hitachi, NEC, NHK, NTT, Nortel, Siemens, and Telia.

FIPA's operations center around annual rounds of specification deliverables. The current specification is available at the FIPA home page ⁷. 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 form the backbone of the FIPA specifications.

4.1 FIPA ACL

FIPA's agent communication language, like KQML, draws 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

⁷ <http://www.fipa.org>

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.

In FIPA ACL, the communication primitives are called communicative acts, or CA's 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.

4.2 FIPA ACL Semantics

SL is the formal language used to define FIPA ACL's semantics. SL is a quantified, multi-modal logic with modal operators for *beliefs* (B), *desires* (D), *uncertain beliefs* (U), and *intentions* (or, *persistent goals*, PG). SL can represent propositions, objects, and actions. We can trace SL's origins to the work of Cohen and Levesque [8] but its current form is primarily based on the work [37]. 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 are 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.

Conformance with the FIPA ACL means that when agent i sends CA a , the FP(a) for i must hold. The unguaranteed RE(a) 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 f . 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

1. holds that the proposition $B_i(f)$ is true ;
2. does not already believe that the receiver has any knowledge of the truth of the proposition $B_i(Bif_j(f) \vee Uif_j(f))$; and
3. intends that the receiving agent should also come to believe that the proposition is true (rational effect $B_j(f)$);

< i , **inform**(j , f) >
 FP: $B_i(f) \wedge \neg B_i(Bif_j(f) \vee Uif_j(f))$
 RE: $B_j(f)$

Fig. 3. FIPA ACL semantics for the communicative act inform. Agent i informs agent j of content f .

5 Comparing KQML and FIPA ACL

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.

Both languages assume a basic non-commitment 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.

Identical syntaxes guarantee that 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, [22] [25] 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 f in the definition of inform with $B_i(f)$ (see Figure 3); still, the two definitions will not be semantically equivalent.

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 CA's 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.

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. FIPA is currently exploring the accommodation of facilitation primitives in FIPA ACL.

In theory, the similarity in basic assumptions and syntax among existing ACLs (under an assumption of shared naming and registration conventions) means that only the communication primitive-specific code 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 definitions⁸. 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 chose should be based on pragmatic concerns.

⁸ When asking programmers about the development of the communication primitive-specific code, the author has often received responses that amount to *we read the natural language definition of the primitives*.

6 Agent Communication Languages: Beyond the Semantics

The emergence of FIPA ACL might have been an additional headache for implementers who must decide for themselves which one of the two ACLs to use. The largely semantic (and subtle) differences between the two ACLs pale in comparison to the more significant issues in terms of the practically useful incorporation of ACLs in agent systems.

In principle, any system that is to use KQML (or FIPA ACL, for that matter) must provide the following things:

- a suite of APIs that facilitate the composition, sending, and receiving of ACL messages;
- 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
- 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 listed under item 2; they ought to exist as a continuous running service available to any new agent.

In practice, the craft of building ACL-capable agents is a bit more complex. First of all, the canonical ACL message syntax (both in FIPA ACL and KQML) further includes additional message parameters whose semantics go beyond that of the primitives. These parameters are unaccounted for in the deep semantics but are essential to the processing of an ACL message. For example: the ACL specification does not specify any conventions for a naming scheme and an associated namespace; or, there is nothing in the message that suggests the method of delivery (protocol). In other words, the ACL includes several pragmatic (i.e., operational) aspects, in addition to what is formally specified at the semantics layer. Such pragmatic aspects are necessary for parsing in and out of the ACL, i.e., digesting and composing well-formed ACL syntax (which is Lisp-like) to extract or insert message parameters or, for queuing (and de-queuing) ACL messages for delivery through TCP or some other network protocol. Further pragmatic issues include the conventions for finding agents and initiating interaction.

Although, in our view, these issues are actually outside of the ACL's scope they are fundamental to an agent's ability to speak an ACL. Research groups that designed and built agent systems had to establish their own conventions and choices for these pragmatic issues. Actually, the various APIs for KQML and FIPA ACL provide nothing (as expected) regarding the actual processing of

ACL messages (depending on the primitive), since respecting the deep semantics of the primitives is the responsibility of the application that makes use of those API's. Such API's today mainly take care of the parsing and queuing tasks mentioned above. Performing these tasks is what using KQML (or FIPA ACL, for that matter) has come to mean. For all intents and purposes, compliance with the ACL's specification means compliance with all these pragmatic conventions. These conventions are not part of the standard (to the extent that the ACL semantics is standardized) and the subtle (or not so subtle) discrepancies amongst their implementations account in large part for the situation today in which there is often a lack of interoperability between systems using the same ACL⁹.

Such issues go beyond the specification of the ACL itself. After all, syntax and semantics suffice to fully specify any language, including an ACL. Still, without solutions to these questions, an ACL is only a useless artifact. FIPA was the first comprehensive effort to address collectively these matters; we outline some of its contributions in Section 9. Next, we introduce the topics that have emerged as fundamental to the construction of interesting systems of communicating agents. Our opinion is that agent communication language standardization, refers to agent communication standardization, which includes issues and considerations that transcend the semantics. In the remaining of this article, we cover in more detail the current work on these issues.

Syntax and Encoding

The core semantics of an ACL is defined as the *deep* semantics (i.e., semantics in the sense of declarative knowledge representation) of its (communication) primitives. This semantics are expressed in some knowledge representation language: SL in the case of FIPA ACL. This semantics only takes into account the speaker, the hearer (in speech act terminology) and the content of the communicative act. The speaker, the hearer and the content correspond to the `:sender`, the `:receiver` and the `:content` of the syntactic representation of the ACL. The canonical syntactic form of the ACL message (for both KQML and FIPA ACL) is a Lisp-like ASCII sequence.

We advocate an abstract syntax for the ACL. Fixed elements of the abstract syntax are the ones that have a direct semantic equivalent (sender, receiver, performative, content). Messages might have multiple encodings, Java objects, Lisp-like ASCII, etc. We advocate XML as a more interoperable and flexible encoding and we elaborate these points in Section 8.

Services and Infrastructure

Every multiagent system that uses an ACL has a homegrown implementation of the APIs that we mentioned at the opening of this section. There are more than

⁹ The differences in sets of primitives used and their intended meaning constitute a second-in-order interoperability barrier that is not confronted due to these more mundane "lower-level" obstacles.

a handful of APIs written in Java, for Java agents and each provides its own infrastructure of basic services. Such API's provide the means to compose well-formed ACL messages, to be able to parse them, to send and receive them over the network and to have some scheme for naming agents, etc. We do not expect a one-size-fits-all solution to these problems. We advocate WWW-like naming schemes and multiple encodings (or, XML encodings, as we hope) as means for reducing the overhead of building an agent that can interact with other, existing agents. An XML encoding will help, for example, with the overhead of composing and parsing because there are XML-processing tools. For facilitators, name servers, *etc.*, things will remain in flux because solutions and approaches are application and domain dependant, but it would be much easier to develop such services if we can build on top of a WWW- like infrastructure of agent naming and message encoding.

Conversations

As we mentioned earlier, the original KQML specification suggested an implicit sequencing of messages in agent interactions. First in [23] and later in [27] [5] [11] the idea of conversations for communicating agents that use an ACL, was introduced and further explored. Conversations mark a shift from individual messages to sequences (exchanges) that agents engage in order to perform certain tasks. The emphasis shifts from the agent's internals to the agent's behavioral patterns. In that sense they might be more useful than the semantics in software development. The body of work in the area has been impressive enough to warrant at least on workshop on conversations for communication agents ¹⁰.

ACL's and the WWW

There was no WWW when KQML first appeared (there were no agents, either, for that matter). KQML and FIPA ACL have evolved at a considerable distance from the mainstream of Internet technologies and standards. No Internet standardization organization has ACL's in their agenda. With the exception of the Artemis project (France Telecom), no major industry player has committed major resources to depend upon, or to develop, ACL's, although there are some plans for future work that will take advantage of FIPA technologies, as they become available. At the same time the WWW is a huge repository of information and agents are almost always referred to in conjunction with the WWW. ACL's are driving a great part of the agent work (FIPA ACL is the centerpiece of the FIPA effort); it is thus reasonable to suggest that ACL work ought to integrate easily with the WWW and to be able to leverage WWW tools and infrastructure. Some first step towards integrating ACL work into the WWW include: abstract syntax for ACL messages, domain specific content languages, a XML encoding for the messages and the content. And all that, need not be at the exclusion of any type of agents, e.g., mobile agents. Of particular interest are

¹⁰ In the Agents-99 conference.

efforts to create content languages that are suitable *Knowledge Representations* for WWW content such as the DARPA Agent Modelling Language (DAML ¹¹) and RuleML ¹².

7 Agent Communication Languages and Conversations

An Agent Communication Language provides agents with a means to exchange information and knowledge. ACLs, such as KQML or FIPA ACL, are languages of propositional attitudes. ACLs are intended to be above the layer of mechanisms such as RPC or RMI because: (1) they handle propositions, rules and actions instead of simple objects (with no semantics associated with them), and (2) the ACL message describes a desired state in a declarative language, rather than a procedure or method. But ACLs by no mean cover the entire spectrum of what agents may want to exchange. More complex *objects* can and should be exchanged between agents, such as shared plans and goals, or even shared experiences and long-term strategies.

As discussed, the ACL itself defines the types of messages (and their meaning) that agents may exchange. Agents though, do not just engage in single message exchanges but they have *conversations*, *i.e.*, task-oriented, shared sequences of messages that they observe, in order to accomplish specific tasks, such as a negotiation or an auction. At the same time, some higher-level conceptualization of the agent's strategies and behaviors drives the agent's communicative (and non-communicative) behavior. When an agent sends a message, it has expectations about how the recipient will respond to the message. Those expectations are not encoded in the message itself; a higher-level structure must be used to encode them. The need for such conversation policies is increasingly recognized by both the KQML [27] and the FIPA communities [15, 12].

A conversation is a pattern of message exchange that two (or more) agents agree to follow in communicating with one another. In effect, a conversation is a pre-arranged coordination protocol. A conversation lends context to the sending and receipt of messages, facilitating interpretation that is more meaningful.

Although, conversations have become part of many infrastructures for ACL-speaking agents, relatively little work has been devoted to the problem of conversation specification and implementation. For conversations to be used for agent coordination, the following three issues require attention:

1. Conversation specification: How can conversations best be described so that they are accessible both to people and to machines?
2. Conversation sharing: How can an agent use a specification standard to describe the conversations in which it is willing to engage, and to learn what conversations are supported by other agents?
3. Conversation aggregation: How can sets of conversations be used as agent API's to describe classes of capabilities that define a particular service?

¹¹ <http://www.daml.org>

¹² <http://www.dfki.de/ruleml>

Well-defined, sharable conversation protocols, with testable, desirable properties, can be used to coordinate agents that attempt to accomplish specific tasks. Typically, a conversation protocol is associated with a specific task, such as *registration*, or a particular type of a *negotiation*. Agents adhering to the same conversation protocol, can coordinate their communicative actions as they attempt to accomplish the task suggested by the conversation protocol. Such a coordination is akin to a scripted interaction, with specific properties, rather than coordinated action resulting from a deep understanding of the domain and the task at hand. Nevertheless, it can be effective for tasks that can be adequately described in the form of possible sequences of communicative interactions.

A specification of a conversation that could be shared among agents must contain several kinds of information about the conversation and about the agents that will use it. First, the sequence of messages must be specified. Traditionally, *deterministic finite-state automata* (DFA's) have been used for this purpose; DFA's can express a variety of behaviors while remaining conceptually simple. For more sophisticated interactions, however, it is desirable to use a formalism with more support for concurrency and verification. Next, the set of roles that agents engaging in a conversation may play must be enumerated, and the constraints and dependencies between individual messages need to be captured. Many conversations will be dialogues, and will specify just two roles; however conversations with more than two roles are equally important, representing the coordination of communication among several agents in pursuit of a single common goal. For some conversations, the set of participants may change during the course of the interaction.

These capabilities will allow the easy specification of individual conversations. To develop systems of conversations though, developers must have the ability to extend existing conversations through specialization and composition. Specialization is the ability to create new versions of a conversation that are more detailed than the original version; it is akin to the idea of subclassing in an object-oriented language. Composition is the ability to combine two conversations into a new, compound conversation. Development of these two capabilities will entail the creation of syntax for expressing a new conversation in terms of existing conversations, and for linking the appropriate pieces of the component conversations.

The set of conversations in which an agent will participate defines an interface to that agent. Thus, standardized sets of conversations can serve as abstract agent interfaces (AAIs), in much the same way that standardized sets of function calls or method invocations serve as APIs in the traditional approach to system-building. That is, an interface to a particular class of service can be specified by identifying a collection of one or more conversations in which the provider of such a service agrees to participate. Any agent that wishes to provide this class of service need only implement the appropriate set of conversations.

Implementing and expressing conversations is not a new idea. As early as 1986, Winograd and Flores [40] used state transition diagrams to describe conversations. The COOL system [3] has perhaps the most detailed current FSM-

based model to describe agent conversations. Each arc in a COOL state transition diagram represents a message transmission, a message receipt, or both. One consequence of this policy is that two different agents must use different automata to engage in the same conversation. Other conversation models have been developed, using various approaches. Extended FSM models, which, like COOL, focus more on expressivity than adherence to a model, include Kuwabara et al. [21], who add inheritance to conversations, Wagner et al. [39], and Elio and Haddadi [13], who defines a multi-level state machine, or Abstract Task Model (ATM). A few others have chosen to stay within the bounds of a DFA, such as Chauhan [7], who uses COOL as the basis for her multi-agent development system¹³, Nodine and Unruh [32], and Pitt and Mamdani [35], who uses DFAs to specify protocols for BDI agents. Also using automata, Martin et al. [29] employs Push-Down Transducers (PDT). Lin et al. [28] and Cost et al. [10] demonstrate the use of CPNs, and Moore [30] applies state charts. Parunak [33] introduces Dooley Graphs. Bradshaw [4] introduces the notion of a conversation suite as a collection of commonly-used conversations known by many agents. Labrou [22] uses definite clause grammars to specify conversations.

While each of these works makes contributions to our general understanding of conversations, more work needs to be done to facilitate the sharing and use of conversation policies by agents. FIPA has contributed to the overall effort by providing specifications for a number of interesting conversation protocols.

8 Agent Communication and the WWW

A number of researchers within and outside FIPA have suggested that ACL messages ought to be encoded in XML, in their entirety, i.e., both the message layer and the content layer should be in XML.

XML is a language for creating markup languages that describe data. XML is a machine-readable and application-independent encoding of a *document*, e.g., of a FIPA ACL message including its content. In contrast to HTML which describes document structure and visual presentation, XML describes data in a human-readable format with no indication of how the data is to be displayed. It is a database-neutral and device-neutral format; data marked up in XML can be targeted to different devices using, for example, *eXtensible Style Language* (XSL). The XML source by itself is not primarily intended directly for human viewing, though it is human-understandable. Rather, the XML is rendered using standard available XML-world tools, then browsed, e.g., using standard Web browsers or specialized other browsers/editors. One leading method for rendering is via XSL, in which one specifies a stylesheet.

XML is a meta-language used to define other domain- or industry-specific languages. To construct a XML language (also called a *vocabulary*), one supplies a specific *Document Type Definition* (DTD), which is essentially a context-free grammar like the Extended BNF (Backus Naur Form) used to describe computer

¹³ More recent work with this project, JAFMAS, explores conversion of policies to standard Petri Nets for analysis [16].

languages. In other words, a DTD provides the rules that define the elements and structure of the new language. For example, if we want to describe employee records, we would define a DTD which states that the <NAME> element consists of three other elements called <FIRST>, <MIDDLE>, and <LAST>, in that order. The DTD would also indicate if any of the nested elements is optional, can be repeated, and/or has a default value. Any browser (or application) having an XML parser could interpret the employee document instance by learning the rules defined by the DTD.

Elsewhere, [19] we have defined ACML (Agent Communication Markup Language), a XML language for describing FIPA ACL message. The deep semantics of the communication primitives in ACML is simply taken to be the same as previously. This semantics is not affected by encoding in XML instead of the previous ASCII; it is defined independently of the choice of syntactic encoding.

Encoding ACL messages in XML offers some advantages that we believe are potentially quite significant.

The XML-encoding is easier to develop parsers for than the Lisp-like encoding. The XML markup provides parsing information more directly. One can use the off-the-shelf tools for parsing XML, instead of writing customized parsers to parse the ACL messages. A change or an enhancement of the ACL syntax does not have to result to a re-writing of the parser. As long as such changes are reflected in the ACL DTD, the XML parser will still be able to handle the XML-encoded ACL message. In short, a significant advantage is that the process of developing or maintaining a parser is much simplified.

More generally, XML-ifying makes ACL more WWW-friendly, which facilitates Software Engineering of agents. Agent development ought to take advantage and build on what the WWW has to offer as a software development environment. XML parsing technology is only one example. Using XML will facilitate the practical integration with a variety of Web technologies. For example, an issue that has been raised in the ACL community is that of addressing security issues, e.g. authentication of agents' identities and encryption of ACL messages, at the ACL layer. The WWW solution is to use certificates and SSL. Using the same approach for agent security considerations seems much simpler and more intuitive than further overloading ACL messages and the ACL infrastructure to accommodate such a task.

As we mentioned earlier, the operational semantics of the pragmatic aspects of ACL can differ subtly between implementations or uses, and there is today a problem practically of interoperability. XML can help with these pragmatics, by riding on standard WWW-world technologies: to facilitate the engineering, and as a by-product to help standardize the operational semantics, thereby helping make interoperability really happen.

Because XML incorporates links into the ACL message, this takes a significant step toward addressing the problem (or representational layer) of specifying and sharing the ontologies used in an ACL message's content. The values of the ACL parameters are not tokens anymore, but links that can point to objects and/or definitions. Although the ontology slot has been present since the in-

ception of ACLs, the ACL community has not been very clear on how this information is to be used by the agent. This vagueness, further compounded by the scarcity of published ontologies, can be addressed by interfacing the ACL message to the knowledge repository that is the WWW.

More generally, links may be useful for a variety of other purposes. For example, the receiver parameter might have a link to network location that provides information about the agent's identity: e.g., its owner, contact and administrative information, communication primitives that the agent understands, network protocols and ports at which it can receive messages, conversation protocols it understands, etc.. This type of information is necessary for establishing an extended interaction with another agent and has to somehow be available to an agent's potential interlocutors. The same argument can be made about the other message parameters.

9 FIPA: Services and Infrastructure

FIPA's approach on the ACL specification is somewhat different than that of KQML, partly because FIPA is an organized body with the authority to take action on its own specifications and the KQML community is a loose network of researchers.

The initial intent was to provide an ACL specification that only covered syntax and semantics of the primitives, carefully avoiding addressing any of the pragmatic considerations at the ACL specification level. Most of the pragmatic considerations were delegated to a different group, charged with Agent Management. The boundaries between the two were often challenged, but with few exceptions (such as conversation specifications been introduced as part of the ACL specification document) the divide persevered. The conflict relates to striving for a balance between a semantically (and theoretically) *pure* ACL specification and the need for a *practical* ACL, from the system-building point of view.

In recognition of the importance of conversations FIPA provides a number of pre-specified protocols. Additionally, FIPA has addressed some of the other issues we mentioned previously, in its Agent Management Transport and Agent Management specifications ¹⁴.

The FIPA Agent Message Transport specifications deal with the delivery and representation of messages across different network transport protocols, including wireline and wireless environments. The agent message transport reference model provides facilities for: (1) general support for a Message Transport Service within an agent platform, (2) guidelines for using specific Message Transport Protocols, such as IIOP, HTTP and WAP and (3) support for encodings that are suitable for each transport protocol, such as an XML encoding for HTTP and a bit-efficient encoding for WAP.

The FIPA Agent Management Specification provides the framework within which FIPA agents exist and operate. It establishes the logical reference model

¹⁴ See [6] for a recent summary of these specifications.

for the creation, registration, location, communication, migration and retirement of agents. The reference model describes the primitives and ontologies necessary to support the following services in an agent platform: (1) white pages, such as agent location, naming and control access services, (2) yellow pages, such as service location and registration services, which are provided by the *Directory Facilitator*.

In the FIPA model agents belong to one or more agent platforms which provide basic services in a way consistent with the above specifications. Sixteen FIPA platforms have been implemented by diverse companies, four of these are freely accessible under open source, that support these specifications. These FIPA platforms have been distributed and tested in large-scale projects, which collectively have been downloaded several thousands of times. For the latest information on FIPA specifications and available FIPA-compliant platforms, the reader is encouraged to check the FIPA homepage ¹⁵.

10 Conclusions

When Knowledge Query and Manipulation Language (KQML) first appeared 10 years ago, it was not an Agent Communication Language (ACL), the term *agents* did not refer to the same kind of entity it refers to today and the KQML specification was a little more than a document combining natural language and syntactic sugaring. Over a period of time KQML has come to define the concept of an ACL and in the process the ACL has become the centerpiece of a large category of agent systems. Almost inevitably an ACL has become a loosely-defined concept that encompasses a variety of issues which may or may not be ACL-relevant depending on one's point of view. The more *conservative* viewpoint advocates that semantics is the one and only real issue. Agent development often suggests though that semantics is the least important concern when one actually builds an agent system. The emergence of FIPA ACL was touted as an attempt for a *cleaner, purer* ACL with well-defined semantics. But aside from the inherent limitations of current ACL semantic approaches, the efforts of many researchers to develop multi-agent systems have brought to the foreground issues and considerations that are at least as important as the semantics for interoperable agent systems; FIPA faced the same problems that the KQML community had been unable to put to rest. The naive (in hindsight) early concept of KQML messages as speech-act-like message types, expressed as ASCII strings, with a LISP-like syntax, that are transported over TCP/IP connections, and aimed at knowledge and information exchange between software systems that are viewed as Virtual Knowledge Bases, exhibited its limitations a long time ago.

Standardization, though, revolves mainly around usability. How to use the semantics in order to develop agent systems that communicate and interoperate using a ACL? Are conversation protocols a more useful tool for agent development than the semantics? Where are the basic services that the ACL presumes

¹⁵ <http://www.fipa.org>

to exist? Do ACL messages have to be ASCII strings encoded in an AI-ish syntax? What do agents actually "talk" about? ACL messages are expected to be opaque to the content language expressions they "carry" but this does not answer the question of which content languages are useful for applications. Is there any connection between agents, ACL's and the World Wide Web? These are the questions we investigated. To some of them we provided (partial) answers and examined the current work, to others we suggested possible avenues of research; our persistent goal, though, was to finally get past the semantics and come to terms with a broader notion of agent communication standardization.

References

1. ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group, July 1993.
2. J. L. Austin. *How To Do Things With Words*. Harvard University Press, second edition, 1962, 1975.
3. Mihai Barbuceanu and Mark S. Fox. COOL: A language for describing coordination in multiagent systems. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 17–25, San Francisco, CA, 1995. MIT Press.
4. Jeffrey M. Bradshaw. KAOs: An open agent architecture supporting reuse, interoperability, and extensibility. In *Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996.
5. Jeffrey M. Bradshaw, Stuart Dutfield, Pete Benoit, and John D. Woolley. Kaos: Toward an industrial-strength open agent architecture. In Jeffrey M. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, 1997.
6. Bernard Burg, Jonathan Dale, and Steven Willmott. Open standards and open source for agent-based systems. *AgentLink*, January 2001.
7. Deepika Chauhan. JAFMAS: A Java-based agent framework for multiagent systems development and implementation. Master's thesis, ECECS Department, University of Cincinnati, 1997.
8. Philip R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2–3):213–361, 1990.
9. Philip R. Cohen and H.J. Levesque. Communicative actions for artificial agents. In *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS'95)*. AAAI Press, June 1995.
10. R. Scott Cost, Ye Chen, Tim Finin, Yannis Labrou, and Yun Peng. Modeling agent conversations with colored petri nets. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 59–66, Seattle, Washington, May 1999.
11. R. Scott Cost, Tim Finin, Yannis Labrou, Xiaocheng Luan, Yun Peng, Ian Soboroff, James Mayfield, and Akram Boughannam. Jackal: A java-based tool for agent development. In *Working Notes of the Workshop on Tools for Developing Agents (AAAI Technical Report)*, Madison, WI, 1998.
12. Ian Dickinson. Agent standards. Technical report, Foundation for Intelligent Physical Agents, October 1997.

13. Renée Elio and Afsaneh Haddadi. On abstract task models and conversation policies. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 89–98, Seattle, Washington, May 1999.
14. Adam Farquhar, Richard Fikes, and James Rice. The Ontolingua server: A tool for collaborative ontology construction. In *KAW96*, November 1996.
15. FIPA. FIPA 97 specification part 2: Agent communication language. Technical report, FIPA - Foundation for Intelligent Physical Agents, October 1997.
16. Alan Galan and Albert Baker. Multi-agent communications in JAFMAS. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 67–70, Seattle, Washington, May 1999.
17. Michael Genesereth and Richard Fikes. Knowledge Interchange Format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, June 1992.
18. Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.
19. B. Grosz and Y. Labrou. An approach to using xml and a rule-based content language with an agent communication language, 1999.
20. Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 2:199–220, 1993.
21. Kazuhiro Kuwabara, Toru Ishida, and Nobuyasu Osato. AgenTalk: Describing multiagent coordination protocols with inheritance. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '95)*, pages 460–465, 1995.
22. Yannis Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland, Baltimore County, August 1996.
23. Yannis Labrou and Tim Finin. A semantics approach for KQML—a general purpose communication language for software agents. In *3rd International Conference on Information and Knowledge Management*, November 1994.
24. Yannis Labrou and Tim Finin. A proposal for a new kqml specification. Technical Report Technical Report TR-CS-97-03, University of Maryland Baltimore County, 1997.
25. Yannis Labrou and Tim Finin. Semantics and conversations for an agent communication language. In Michael Huhns and Munindar Singh, editors, *Readings in Agents*. Morgan Kaufmann, 1997. Reprint of a paper from the Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997 (IJCAI-97).
26. Yannis Labrou, Tim Finin, and Yun Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52, / 1999.
27. Yannis Labrou and Timothy Finin. Semantics and conversations for an agent communication language. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, August 1997.
28. Fuhua Lin, Douglas H. Norrie, Weiming Shen, and Rob Kremer. Schema-based approach to specifying conversation policies. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies, Third International Conference on Autonomous Agents*, pages 71–78, Seattle, Washington, May 1999.
29. Francisco Martin, Enric Plaza, and Juan Rodríguez-Aguilar. Conversation protocols: Modeling and implementing conversations in agent-based systems. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 49–58, Seattle, Washington, May 1999.

30. Scott Moore. On conversation policies and the need for exceptions. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 19–28, Seattle, Washington, May 1999.
31. R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, Fall 1991.
32. M. H. Nodine and A. Unruh. Facilitating open communication in agent systems: the InfoSleuth infrastructure. In Michael Wooldridge, Munindar Singh, and Anand Rao, editors, *Intelligent Agents Volume IV – Proceedings of the 1997 Workshop on Agent Theories, Architectures and Languages*, volume 1365 of *Lecture Notes in Artificial Intelligence*, pages 281–295. Springer-Verlag, Berlin, 1997.
33. H. Van Dyke Parunak. Visualizing agent conversations: Using enhanced dooley graphs for agent design and analysis. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS '96)*, 1996.
34. Ramesh S. Patil, Richard E. Fikes, Peter F. Patel-Schneider, Don McKay, Tim Finin, Thomas Gruber, and Robert Neches. The darpa knowledge sharing effort: Progress report. In Michael Huhns and Munindar Singh, editors, *Readings in Agents*. Morgan Kaufmann Publishers, 1997. (reprint of KR-92 paper).
35. Jeremy Pitt and Abe Mamdani. Communication protocols in multi-agent systems. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 39–48, Seattle, Washington, May 1999.
36. Jeremy Pitt and Abe Mamdani. A protocol-based semantics for an agent communication language. In *IJCAI*, pages 486–491, 1999.
37. M.D. Sadek. A study in the logic of intention. In *Proceedings of the 3rd Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 462–473, Cambridge, MA, 1992.
38. Ira A. Smith and Philip R. Cohen. Toward a semantics for an agent communications language based on speech-acts. In *Proceedings of the 13th National Conference on Artificial Intelligence*. AAAI/MIT Press, August 1996.
39. Thomas Wagner, Brett Benyo, Victor Lesser, and Ping Xuan. Investigating interactions between agent conversations and agent control components. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 79–88, Seattle, Washington, May 1999.
40. Terry Winograd and Fernando Flores. *Understanding Computers and Cognition*. Addison-Wesley, 1986.