The current landscape of Agent Communication Languages

Yannis Labrou, Tim Finin and Yun Peng
LABORATORY for ADVANCED INFORMATION TECHNOLOGY
Computer Science and Electrical Engineering Department,
University of Maryland, Baltimore County,
{jklabrou,finin,ypeng}@cs.umbc.edu
Phone: (410) 455 3624 Fax:(410) 455 3969

Abstract

Despite the substantial number of multi-agent systems that use an Agent Communication Language (ACL) the dust has not settled yet over the landscape of ACLs. The semantic specification issues have monopolized the debate at the expense of other important pragmatic issues that must be adequately resolved in the immediate future if ACLs are going to support the development of robust agent systems. After introducing some of the basic concepts relating to Agent Communication Languages, we cover KQML and FIPA ACL, the two existing fully-specified ACLs. We give a brief introduction to their semantics and the issues relating to semantic descriptions of ACLs. We then shift our focus beyond the semantics and point to problems and limitations shared by both ACLs. Questions such as the nature of conformance of an agent system with an ACL specification and issues such as naming, registration, authentication, basic facilitation services, etc.*, may or may not be (technically speaking) part of an ACL specification, but we feel that the answers and solutions to such problems can "make or break" an ACL. We finally discuss the future of ACL standardization efforts and identify the issues that are likely to emerge as we gain experience in building and deploying agent-based systems.*

Final draft of a paper to appear in IEEE Intelligent Systems*, volume 14, number 2, March/April, 1999.*

1 Introduction

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, I, 4

Although the first primates that could be considered our remote ancestors appeared 1 to 2 million years ago (depending on who you ask and the latest findings), it has only been less than 50,000 years since our species "invented" what we would recognize today as a language. In that short time (evolutionary) we managed to raise ourselves to a level unparalleled by other creatures on our planet. Admittedly, the attribution of such dramatic advancement solely to language is not something that can 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 but beyond facilitating the day-to-day business, language "granted" memory to communities. The experiences and acquired knowledge of its members could be conveyed to the next generation.

For a language to be a *language*, it does not have to be a spoken, "natural language" like English, or Greek, or Mandarin Chinese. The signs that baseball managers, coaches and players use, or the sign language for the deaf are examples of *languages* that exhibit a fundamental property of useful languages: the meaning of their tokens are shared. This is not to say that one can not have a private language, whose symbols are only understood to herself; simply, there is not much you can do with such a language. If this private language becomes public, well now "we are talking." Which leads us to a fundamental characteristic of any language, *i.e.*, 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 the role for software agents. Agents [1] [3, 16] is suggest a paradigm for software development which emphasizes autonomy (both at design time and run time), adaptivity (change is everywhere) and cooperativity (humans agents do it all the time, so should the software ones, too). This approach seems appealing in a world of distributed, heterogeneous systems. Languages for communicating agents promise to play the role that language played for their human counterparts. Agents need an Agent Communication Language (ACL) in order to interact in a shared language, hiding the details of their internals and to build communities of agents that can tackle problems that no individual agent can.

After introducing some concepts useful in discussing Agent Communication Languages [2] (Section 2) we trace the emergence and the context of ACLs (Section 3) giving us the opportunity to re-iterate the relevance of (content) representation languages and ontologies to ACLs. KQML is our vehicle for introducing the fundamental notions of an ACL (Section 4). The semantics of KQML have been the single most important debate issue over the years and we include a brief overview (Section 4.1). The Foundation for Intelligent Physical Agents (FIPA) is the first organized effort focusing on developing standards in the broader area of agents; after introducing FIPA (Section 5) we discuss FIPA ACL (Section 5.1). Since FIPA ACL follows the same basic concepts with KQML we give a flavor of its semantics which is the major point of difference with respect to KQML. As we attempt a comparative evaluation of KQML and FIPA ACL (Section 6) we try to go beyond the dominant issue of semantics and point to problems and limitations that both ACLs have and ought to be addressed. While in this forward-looking mood we take a look at a few of the many systems that have used an ACL in the past, emphasizing the aspects that manifest the areas and issues of immediate concern and future work (Section 7). Before concluding we discuss the future of standardization efforts in the area of ACLs and the issues that emerge as central as ACLs get used by systems (Section 8).

2 Basic concepts of an Agent Communication Language

An ACL provides agents with a means to exchange information and knowledge; Genesereth has gone as far as equating agency with the ability of a system to exchange knowledge using an ACL [13]. Of course other means and approaches have been used over the years to achieve the lofty goal of seamless exchange of information and knowledge between applications. From Remote Procedure Call (RPC) or Remote Method Invocation (RMI), to CORBA and Object Request Brokers (ORB's) the goal has been the same. What distinguishes ACLs from past such efforts are the *objects of discourse* and their semantic complexity. ACLs like KQML or FIPA ACL, stand a level above CORBA, because: (1) they handle propositions, rules and actions instead of simple objects (with no semantics associated with them), and (2) the ACL message describe a desired state in a declarative language, rather than a procedure or method. But ACLs by no mean cover the entire spectrum of what applications 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.

At the technical level, when using an ACL, agents transport messages over the network using some lower-level protocol (SMTP, TCP/IP, IIOP, HTTP, etc.). 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 follow, 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.

Traditionally, we understand the message types of ACLs as *speech acts*, which in turn are usually accounted for in terms of beliefs, desires, intentions and similar modalities. This kind of intentional-level description can either be just a useful way to view a system or it can have a concrete computational aspect. The latter case describes a large range of BDI[3] agents which 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 knowledge, goals and commitments of an agents, commonly known as a BDI theory. Despite the criticism that BDI theories and BDI agents have faced, such as the number and choice of modalities and the fact that multi-modal BDI logics have neither complete axiomatizations nor they are efficiently computable, they offer an appealing framework to account for agent communications, because agents, when communicating, they communicate their BDI states and/or attempt to alter their interlocutors BDI states.

3 The origin of ACLs

The Knowledge Sharing Effort [21, 24] (KSE) was initiated circa 1990 by DARPA and it enjoyed the participa-

[1] We will always mean *software agents* when mentioning *agents*.

[2] We will use ACL as the abbreviation for Agent Communication Language, when referring to an ACL as a concept or to ACLs collectively. There is an ACL simply named *ACL* but the context will hopefully disambiguate what we mean.

[3] BDI stands for *Belief*, *Desire* (or Goal) and *Intention*.

tion of dozens of researchers from both academia and industry. Its goal was to develop techniques, methodologies and software tools for *knowledge sharing and knowledge 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 KSE focused on defining that common language. In the KSE model, software systems are viewed as (virtual) knowledge bases that exchange propositions using a language that expresses various complex attitudes [4] interest in, *etc.*) about these propositions.

Although originally agents were not part of the KSE vocabulary the conceptual break-down 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, the first layer is that of (syntactic) translation between languages in the same family (or between families) of languages [5]. An other layer is concerned with guaranteeing that the semantic content of tokens is preserved among applications; in other words, the same concept, object, or entity has a uniform meaning across applications even if different "names" are used 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 that hold among them. An ontology consists of terms, their definitions, and axioms relating them [15]; 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 ask other agents, to inform them, to request their services for a task, to find other agents who can assist them, to monitor values and objects, and so on. Such functionality, in an open environment, can not be provided by a simple Remote Procedure Call (RPC) mechanism. Agents issue requests by specifying not a procedure but a desired state in a declarative language, *i.e.*, in some Agent Communication Language.

Within the KSE, these layers were viewed as independent of another. The ACL is only concerned with capturing propositional attitudes, regardless of how propositions are expressed. But still, propositions are what agents will be "talking" about. KIF, [12] a particular logic language, was proposed within the KSE as a standard to use to describe things within computer systems, *e.g.*, expert systems, databases, intelligent agents, etc. Moreover, it was specifically designed, within the context of the KSE, to make it useful as an "interlingua". By this we mean a language which is useful as a mediator in the translation of other languages. KIF is a prefix version of first order predicate calculus with extensions to support non-monotonic reasoning and definitions. The language description includes both a specification for its syntax and one for its semantics. Ontolingua [11] and a variety of supporting tools, was 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 achieving consensus on common shared ontologies by geographically distributed groups. These tools are built around Ontolingua, a language designed for describing ontologies with it, and make use of the world-wide web to enable wide access and provide users with the ability to 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 back to the library.

4 KQML and ACL concepts

KQML [1, 18] illustrates the basic concepts of existing ACLs. "Existing ACLs" are KQML with its many dialects and variants, and FIPA ACL. With the exception of *ACL*, a KQML variant that assumes KIF as the content language, all KQML dialects and FIPA ACL follow the same 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. So, KQML is independent of the transport mechanism (TCP/IP, SMTP, IIOP, *etc.*), independent of the content language (KIF, SQL, STEP, Prolog, *etc.*) and independent of the ontology assumed by the content.

The KQML language is divided into three layers: the content layer, the message layer, and the communication layer. The content layer bears the actual content of the message, in the programs own representation language. KQML can carry any representation language, including languages expressed as ASCII strings and those expressed using a binary notation. Every KQML implementation ignores the content portion of the message, except to determine where it ends. The communication level encodes a set of features to

[4] The proper term is *propositional attitudes*. Propositional attitudes are three-part relationships between: (1) an agent, (2) a content-bearing proposition (*e.g.*, "it is raining"), and (3) a finite set of propositional attitudes an agent might have with respect to the proposition (*e.g.*, believing, asserting, fearing, wondering, hoping, *etc.*). For example, $< a, fear, raining(t_{now}) >$.

[5] The Object Management Group (OMG) standardization effort is an example of work in this direction, within the family of object-oriented languages.

the message which describe the lower level communication parameters, such as the identity of the sender and recipient, and a unique identifier associated with the communication. It is the message layer that is used to encode a message that one application would like to transmit to another. The message layer forms the core of the KQML language, and determines the kinds of interactions one can have with a KQML-speaking agent. The primary function of the message layer is to identify the network protocol to be used to deliver the message and to supply a speech act or performative which the sender attaches to the content (such as that it is an assertion, a query, a command, or any of a set of known *performatives*[6]). In addition, since the content is opaque to KQML, this layer also includes optional features which 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 even though their content is inaccessible.

The syntax of KQML is based on the familiar *s-expression* used in Lisp, i.e., 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:

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

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`, `:receiver` keywords form the communication layer and the performative name, with the `:language` and `:ontology` form the message layer. In due time, *stock-server* might send to *joe* the following KQML message:

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

Though there is 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 set is extensible; a community of agents may 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 variety of interesting agent architectures. Thus, KQML introduces a small number of KQML performatives which are used by agents to describe the meta-data specifying the information requirements and capabilities; KQML also introduces a special class of agents called *communication facilitators* [13]. A facilitator is an agent that performs various useful communication services, *e.g.* maintaining a registry of service names, forwarding messages to named services, routing messages based on content, providing "matchmaking" between information providers and clients, and providing mediation and translation services.

4.1 Semantics for KQML

During the first few years of use, KQML existed with only an informal and partial semantic description. This was identified as one of its shortcomings [8]. During the past few years, several efforts to provide a formal semantics have been put forth.

In [17, 19, 20] the semantics of KQML are provided in terms of *preconditions*, *postconditions* and *completion conditions* for each performative. Assuming a sender A and a receiver B, the preconditions indicate the necessary state for an agent in order 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 a *error* or *sorry* will be the most likely response. Postconditions that describe the states of both interlocutors after the *successful* utterance of a performative (by the sender) and after the receipt and process-

[6] KQML as 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 he is doing so. In English such utterances typically appear in a first-person, present tense declarative form, often accompanied by "hereby", as in "I hereby request you to turn on the computer." Cohen [8] has argued that this is a a poor term to use for all ACL primitive message types, since not all can be construed as actions that the sender can make so just by the sending. We continue to use the term for KQML for historical reasons.

ing (but before a counter utterance) of a message (by the receiver). The postconditions (**Post(A)** and **Post(B)**, respectively) 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 possibly a conversation has taken place and the intention associated with the performative that started the conversation, has been fulfilled.

Establishing the preconditions of 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 are taken to 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 a message 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 they can be combined to compose agents' states.

The following is an example of semantics for sender A and receiver B in this framework

$tell(A, B, X)$

- **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))

This semantics for *tell* suggest that an agent can not offer unsolicited information to some other agent. This can be easily amended by introducing another performative, let us call it *proactive-tell* which has the same semantic description as *tell* but with the **Pre(A)** being BEL(A,X), and an empty **Pre(B)**.

The other semantic approach [8, 27] builds on earlier work on defining rational agency [9]. In this body of work, the term *performative* is deemed inappropriate to describe KQML's communication primitives and the suggested approach views the language's reserved message types as *attempts* for communication. These attempts involve two (or more) rational agents that (temporarily) form "teams" in order to engage in communication. This approach strongly links the ACL semantics to the agent theory assumed for the agents involved in an ACL exchange.

5 The Foundation for Intelligent Physical Agents

The Foundation for Intelligent Physical Agents (FIPA) is a non-profit 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 the above description suggests, FIPA is a standards organization in the area of software agents. The often confusing "physical" adjective in its name was originally intended to cover agents of the robotic variety but over time the adjective's presence has come to serve as a reminder that physical, *i.e.*, human agents and interaction with them is part of the association's scope.

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

FIPA's operations are built around annual rounds of specification deliverables. Current specification is FIPA97 and can be found at the FIPA home-page [7] The *modus operandi* of FIPA is to assign tasks to Technical Committees (TCs) that have the primary responsibility for producing, maintaining and updating the specifications that are applicable to their tasks. The TC that is most important in the scope of this paper is the TC charged with producing a specification for an Agent Communication Language. Along with the TC in charge of Agent Management (agent services, such as facilitation, registration and agent platforms) and Agent/Software Interaction (integration of agents with legacy software applications) they form the backbone of the FIPA specifications.

5.1 The FIPA Agent Communication Language and its semantics

The FIPA Agent Communication Language (FIPA ACL), 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 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. Every communicative act is described with both a narrative form and a formal semantics based on modal logic. The specification also provides the normative description of a set of high-level interaction

[7] which was `http://drogo.cselt.stet.it/fipa` at the writing of this paper; a new official FIPA page is under construction, at `http://www.fipa.org`

protocols, including requesting an action, contract net and several kinds of auctions.

The FIPA ACL is superficially similar to KQML. Its syntax is identical to that of KQML's except for the different names for some reserved primitives. Thus, it maintains the KQML approach of separating the outer "language" that defines the intended meaning of the message and the inner language, or "content language" that denotes the expression towards which the beliefs, desires and intentions of the interlocutors, as described by the meaning of the communication primitive, apply. The FIPA ACL specification document claims that FIPA ACL (like KQML) does not make any commitment to a particular content language. Although such a claim holds true for most primitives, there are FIPA ACL primitives for which some understanding of the language SL (Semantic Language) is necessary for the receiving agent to understand and process the primitive. [8] We will discuss this important point later.

SL, standing for *Semantic Language* [**?**], is the formal language used to define the semantics of the FIPA ACL. SL is a quantified, multi-modal logic with modal operators for beliefs (B), desires (D), uncertain beliefs (U) and intentions (persistent goals, PG). SL can represent propositions, objects and actions. Its origins can be traced to the work of Cohen and Levesque [9], but its current form is primarily based on the work of Sadek [26]. A detailed description of SL, including its own semantics, is outside the scope of this paper and can be found in the FIPA ACL specification.

The semantics of each communicative act (CA) are specified as sets of SL formulae that describe the act's *feasibility preconditions* and the *rational effect*. For a given CA a, the feasibility preconditions $FP(a)$ describe the conditions that have to hold for the sender of the CA. That is, in order for an agent to properly perform the communicative act a by sending a particular message, the feasibility preconditions must hold for the sender. Note that the agent is not required to obliged to perform a if $FP(a)$ holds, merely that it can, if it chooses. An communicative act's *rational effect* represents the effect that an agent can expect to occur as a result of performing the action and typically specified conditions which should hold true of the recipient. Note that the receiving agent is not required to ensure that the expected effect comes about and may indeed find it impossible. Thus an agent may use its knowledge of the rational effect in order to plan what communicative act to perform, but it is not entitled to automatically assume that the rational effect necessarily holds.

Conformance with the FIPA ACL should be taken to mean that when agent A send CA x, the $FP(x)$ for a should hold; the not to be guaranteed $RE(x)$ is irrelevant to the conformance issue.

This introduction should be enough for a basic understanding of the following example which shows the specification of the communicative act $inform$ in which agent i informs agent j of content ϕ.

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

The content of $inform$ is a proposition and it's 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 ($\neg B_i(Bif_j(\phi) \vee Uif_j(\phi)$).
- intends that the receiving agent also comes to believe that the proposition is true (this is the rational effect $B_j(\phi)$);

6 Comparing ACLs

KQML and FIPA ACL are almost identical with respect to their basic concepts and the principles they observe and differ primarily in the details of their semantic frameworks. In one sense this difference is substantial, resulting in an impossibility of coming up with exact mappings or transformation between KQML performatives and their completely equivalent FIPA primitives, or vice versa. On the other hand, the ineluctable differences that remain might be of little importance to many agents' programmers, if their agents are not true BDI agents. We will elaborate on this argument in Section 8.

Both languages make the same basic assumption of a non-commitment to a reserved content language. In the FIPA ACL case though, some (limited) understanding of SL is necessary to properly process a received message (as in the case of the $request$ CA). The two languages have the same syntax, *i.e.*, a KQML message and a FIPA ACL message, will look syntactically identical, except of course for the different names they use for the communication primitives. This is an important attribute of FIPA ACL. [9] Since 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

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

[9] Originally, FIPA ACL employed a different Prolog-ish syntax; the syntax was changed to KQML's syntax to facilitate the transition of KQML systems to FIPA ACL

network using some 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. If you follow the KQML semantics described in [17], semantically they differ at the level of what constitutes the semantic description (preconditions, postconditions and completion conditions for KQML and feasibility preconditions and rational effect for FIPA ACL) and also, at the level of the choice and definitions of the modalities they employ (the language used to describe agents' states). Although one 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 the FIPA ACL which approximates KQML's tell, you can replace ϕ in the definition of $inform$ with $B_i\phi$.

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 are treated as first class objects in KQML. FIPA ACL, seeking a more pure ACL, does not consider these tasks to be communication primitives (CAs) in their own right and they are treated instead as requests for action; FIPA ACL, in turn, defines a range of reserved actions that cover the registration and life-cycles 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 to have in FIPA ACL the facilitation primitives (*broker*, *recommend* and *recruit*) that they are accustomed to from KQML. Users' requests serve as a sobering reminder of the truth that for an ACL to be practical a careful mix of theoretic and pragmatic considerations is needed.

The emergence of FIPA ACL might be a additional headache for implementors who have to decide for themselves which one of the two ACLs to use. Our answer is bound to cause more headaches. For a system to use KQML (or FIPA ACL for that matter) the following things have to be provided:

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. the code that for every reserved message type (performative or communicative act) 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 that uses the ACL.

Normally as a programmer you would only have to provide (3). Items (1) and (2) should be re-usable components that you can just integrate into your application code; actually (2) does not even need to be integrated because it ought to be provided as continuous running services that a new agent can just use. The sad truth of standardization in ACL languages is that these services have not been the focus of the standardization efforts. There is no service where one can register an agent by just sending a registration message. The disagreement on the issue of such services has resulted (in part) to a multitude of APIs. [10] Every multi-agent system that uses an ACL has a home-grown 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 multi-modal logics that are often non-computable and/or have no efficient implementation. The process of grounding the theory into code will result in a system that differs substantially (and probably unpredictably) from the theory (in paper) that it follows. To make matters worse, if the code does not implement at all the modalities assumed by the semantics, the programmer will most likely follow his (or her) intuitive understanding of the semantics of the communication primitives.

The similarity in basic assumptions and syntax, should (in theory) mean that only (3) ought to change depending of your choice of an ACL. And even then, much to the dismay of those involved in defining the semantics of ACLs, the implementors' 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 should be based on pragmatic concerns.

7 Features of ACL-supporting systems and applications

Over the past few years, a multitude of applications and systems have appeared that are build around an agent communication languages. Instead of providing a compendium

[10] The other reasons are (a) the minor syntactic differences between the various KQML "interpretations" and (b) the different naming schemes employed by the various KQML-speaking agent systems.

of such systems we will focus on a few that give us the opportunity to discuss features and characteristics that exemplify current approaches and trends. The systems we will be discussing belong in one of the following two categories: (a) applications, *i.e.* multi-agent systems that use an ACL for inter-agent communication, and (b) APIs that facilitate the incorporation of ACL-speaking capabilities into an application. Since the ACL itself is an abstraction, *i.e.* a collection of communication primitives that are deemed useful for higher level communication between agents, there is no such thing as an "implementation" of an ACL.

Infosleuth [2, 22] is a project by MCC that emphasizes the semantic integration of heterogeneous information in an open dynamic environment. The communicating agents [11] , primarily written in Java, are supported by 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 that assists with the semantic integration of the handled information. Infosleuth agents engage in conversations rather than single-message exchanges. Knowledgeable Agent oriented System (KAoS) [4] is a Boeing project aimed at providing an infrastructure for agent development. It relies heavily on object oriented technology (using for example a CORBA-based message delivery mechanism) and 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. Agents are designed supporting specialized suites of interactions. Infomaster [14] is an information integration system (from Stanford) that is using *ACL* as its ACL. *ACL* is KQML with KIF as the 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 (JATLite) is a package of Java programs, developed at Stanford, that allow users to quickly create communicating agents. Agents run as applets launched from a browser and for that reason all agents register with an Agent Message Router facilitator (AMR) that handles message delivery. The Java-based Agent Framework for Multi-Agent Systems (JAFMAS) [5, 6] is a set of classes to support implementing communicating agents in Java; it was developed at the University of Cincinnati. JAFMAS supports directed (point-to-point) communication as well as subject-based broadcast communications. The JAFMAS environment, which has support for conversations, is used in AARIA [23], a manufacturing planning and scheduling project. Jackal [10], developed at UMBC, is another Java package that allows applications written in Java to communicate via an ACL (KQML is currently the ACL of choice, but FIPA ACL could be easily supported). Jackal is being used in the CIIMPLEX project [25, 7] which involves manufacturing planning and scheduling. Jackal strongly emphasizes conversations between agents and provides a flexible framework for designing agents around conversations and includes extensive support for registration, naming and control of agents.

We would like to focus on the following characteristics of these systems:

- 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 in doing so with object-oriented languages like Java. This raises again the problem of existing semantic approaches and the conformance problem.
- Many of the new API offer support for conversations. Conversations offer an intuitive manner to structure an agent's activities. Also, given the problematic nature of compliance with the ACL's semantic account, conversations shift the focus from the internals of the agent to its observable behavior (sequences of messages it sends to other agents). Agents can agree on a conversation protocol for a particular task (negotiation or auction) 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 engage in a scripted, prespecified communicative behavior.
- Every single implementation introduces its own variety of supporting agents and services, for tasks such as naming, authentication, monitoring and brokering. Some agreement is needed on the assumptions of these services so that such services can be provided as a standard suite of tools.

8 The future of ACLs and the important issues

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 but FIPA ACL does. At the same time, KQML development model, based on experimentation and continuous feedback from its community of users has helped KQML grow with an emphasis on practical concerns of agent development. The FIPA ACL does not have a community of users yet, as no FIPA ACL applications have appeared, and thus FIPA ACL

[11] The ACL is KQML. All the systems we mention here, are using some variant of KQML as their ACL. As of the spring of 1998 there were no published, deployed systems claiming to use the FIPA ACL.

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 U.S. 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 the ones outlined here) is as thorny as ever [28] and it puts into question the degree of usefulness of such semantic accounts. In the near future the 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 will be of particular interest to these programmers that have no affinity for the standard BDI approach. All of these pragmatic issues are very important for the deployment of agent applications. Their availability will reduce the overhead of agent development and will allow for the shift of focus to what agents *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 these issues but the process is still at an early stage and it remains to be seen to what extend it incorporates the experiences and lessons from dealing with these issues.

Agents are usually mentioned within the context of the WWW and the Internet is the arena in which they are expected to compete. But KQML and FIPA ACL have followed their path away from the mainstream of Internet technologies and standards. No major player has a manifested interest in ACLs, no Internet standardization organization has ACLs in their agenda. Should existing ACLs become more Internet-friendly? And if so, how? Taking advantage of XML (and possibly 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."

9 Conclusions

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 as well as 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.

The KQML language was among the first such ACL to be developed and used. Moreover, is the only one which has enjoyed substantial use by more than its developers to date. However, after eight years of experimentation and experience there are still serious signs of immaturity: (1) in general, different KQML implementations can not interoperate; (2) there is no fixed specification sanctioned by some consensus-creating body; and (3) there is no agreed-upon semantics foundation. Is the KQML experiment a failure?

We think not. KQML has played a large role in defining what an Agent Communication Language 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 to do so. Agent-based systems research is still at an early stage of development, and there has been no benefit to individual research groups in focusing on the interoperability issues. Although the lack of a sanctioned specification has probably impeded the adoption of KQML for many big projects, it has had the benefit of allowing 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 as long as the problem of defining and identifying conformance to the semantics, is not resolved. But given the possibility that it might be impossible to find a satisfactory solution to the latter problem, we will be left with a justifiable sense of disappointment, as computer scientists, for a language that lacks formal, verifiable semantics. However, this disappointment need not touch the programmers who want register their agents, find other agents, to 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 the semantic clarity and purity is partly responsible for the slighting of these crucial issues. After all, what could 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] ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group, July 1993.

[2] R. J. Bayardo, W. Bohrer, A. Cichocki, J. J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk., G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz., R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. Infosleuth: Agent-based semantic integration of information in open and dynamic systems. In *SigMod*, 1997.

[3] Jeffrey Bradshaw, editor. *Software Agents*. AAAI Press/The MIT Press, 1997.

[4] 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.

[5] Deepika Chauhan. *JAFMAS: A Java-Based Agent Framework for Multiagent Systems Development and Implementation*. M.s., University of Cincinnati, 1997.

[6] Deepika Chauhan and Albert Baker. JAFMAS: A multiagent application development system. In Michael Wooldridge and Tim Finin, editors, *Proceedings of the second ACM Conference on Autonomous Agents*, 1998.

[7] Bill Chu, W. Tolone, J. Long, R. Willhelm, Y. Peng, T. Finin, and M. Mathews. Toward intelligent integrated manufacturing planning-execution. *The International Journal of Agile Manufacturing*, 1998.

[8] 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.

[9] P.R. Cohen and H.J. Levesque. Persistence, intention, and commitment. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 33–69. MIT Press, Cambridge, MA, 1990.

[10] 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 Papers of the AAAI-98 Workshop on Software Tools for Developing Agents*. AAAI Press, july 1998.

[11] Adam Farquhar, Richard Fikes, and James Rice. The ontolingua server: A tool for collaborative ontology construction. Technical Report KSL-96-26,, Stanford Knowledge Systems Laboratory, 1996 1996.

[12] M. Genesereth and R. Fikes et. al. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical report, Computer Science Department, Stanford University, 1992.

[13] Michael R. Genesereth and Steven P. Katchpel. Software agents. *CACM*, 37(7):48–53, 147, 1994.

[14] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An information integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1997.

[15] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 2:199–220, 1993.

[16] Michael Huhns and Munindar Singh, editors. *Readings in Agents*. Morgan Kaufmann, 1997.

[17] Yannis Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland, Baltimore County, August 1996.

[18] 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.

[19] 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).

[20] Yannis Labrou and Tim Finin. Semantics for an agent communication language. In Michael Wooldridge, Joerg P. Muller, and Milind Tambe, editors, *Agent Theories, Architectures and Languages IV*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1998.

[21] 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.

[22] Marian Nodine and Amy Unruh. Facilitating open communication in agent systems: The infosleuth infrastructure. In M. Singh, A. Rao, and M. Woolridge,

editors, *4th International Workshop on Agent Theories, Architectures, and Languages*, Providence, RI, 1997.

[23] H.V.D. Parunak, A.D. Baker, and S.J. Clark. The AARIA agent architecture: An example of requirements-driven agent-based system design. In *Proceeding of the First International Conference on Autonomous Agents (ICAA'97)*, pages 482–483, February 1997.

[24] 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).

[25] Yun Peng, Tim Finin, Yannis Labrou, Bill Chu, J.Long, William Tolone, and Akram Boughannam. A multi-agent system for enterprise integration". *Journal of Applied Artificial Intelligence*, 1998.

[26] 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.

[27] 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.

[28] Michael Wooldridge. Verifiable semantics for agent communication languages. In *International Conference on Multi-Agent Systems (ICMAS'98)*, Paris, France, 1998.